IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Order-preserving Languages for the Supervisory Control of Automated Manufacturing Systems

ANAS NOORULDEEN[1], KLAUS WERNER SCHMIDT[2]

[1]Department of Electronics and Communication Engineering, Çankaya University, Ankara, 06790 Turkey (e-mail: anasnooruldeen@gmail.com)
[2]Department of Electrical and Electronics Engineering, Middle East Technical University, Ankara, 06800 Turkey (e-mail: schmidt@metu.edu.tr)

Corresponding author: Klaus Werner Schmidt (e-mail: schmidt@metu.edu.tr).

"This work was supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK)"

ABSTRACT Automated manufacturing systems (AMSs) consist of computer-controlled interconnected manufacturing components (MCs) that are used to transport and process different product types. Each product type requires a certain sequence of processing steps in different MCs. Hereby, multiple product types can share processing steps on the same MC and the paths of different products types can overlap.
In this paper we consider the modeling of AMSs in the scope of supervisory control for discrete event systems (DES). On the one hand, a suitable AMS model must allow the representation of sequential and concurrent processing steps in MCs. On the other hand, such model must be able to track different product types traveling through the AMS so as to process the products correctly. While previous work is commonly concerned with the first requirement, this paper identifies that the existing literature lacks a general treatment of the second requirement. Accordingly, we first introduce order-preserving (OP) languages that preserve the order of different product types in MCs and we propose a suitable finite state automaton model for OP languages. Then, we show that the composition of OP languages again leads to an OP language. That is, modeling MCs by OP languages, an OP model of a complete AMS that is suitable for supervisory control is obtained. In addition, it is possible to use both OP models and non-OP models for general AMSs, where MCs have different properties. We demonstrate the applicability of the proposed modeling technique by a flexible manufacturing system example.

INDEX TERMS Discrete-event systems, automated manufacturing systems, supervisory control, order-preserving models.

I. INTRODUCTION

DISCRETE event system (DES) models are suitable for representing sequential and concurrent processes in technical systems such as automated manufacturing systems (AMSs) [1]–[8]. Such systems consist of a given hardware setup with various manufacturing components (MCs) such as machines, robots and conveyor belts [9]–[13]. It is generally desired to move products along pre-defined paths and to use pre-specified MCs for processing these products in a given sequence while avoiding deadlock situations [14]–[16]. Hereby, it is the case that multiple products are present in different parts of such AMS simultaneously and are processed concurrently. Moreover, advanced AMSs such as flexible manufacturing systems (FMSs) or reconfigurable manufacturing systems (RMSs) are able to process different product types using the same MCs [9], [17]–[20]. Accordingly, multiple products can share (parts of) the same path through the AMS, while requiring different processing steps. In order to process products correctly, it is hence important to keep track of the type of products traveling through such AMS. The main focus of this paper is the modeling and supervisory control of AMSs with different product types and overlapping product paths.

Using DES models, the logic control of AMSs can be carried out in the framework of supervisory control for DES [21], [22]. That is, a supervisor is designed based on a formal model of the AMS and a formal specification of the desired manufacturing sequences. Hereby, achieving the desired system behavior depends both on the adequate formulation of model and specification and the application of a compu-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

tationally efficient supervisor design procedure. Regarding supervisor design, the existing literature considers various aspects of the logic control of AMSs using different modeling frameworks. The avoidance of deadlocks or forbidden states in Petri Nets is addressed in [5], [23]–[28], the enforcement of linear constraints on Petri Net markings is pursued in [29]–[32] and [14], [15], [33]–[35] develop modular and hierarchical methods for the efficient computation of supervisors for AMSs based on finite state automata models.

In the scope of this paper, we consider DES models based on finite state automata. Here, the literature generally adopts models of plant and specification that are composed of multiple components to address the state space explosion problem [33], [36]–[40]. It has to be emphasized that the process of obtaining accurate DES models is not trivial [38], [40]. In particular, tools for supporting the modeling process of both plant and specification including the level of abstraction and the modeling of standard operations are highly desirable. This topic is explored by providing templates for basic DES operations in [41]. Furthermore, recent work such as [16], [42] proposes the concept of distinguishers in order to obtain smaller and more easily understandable specifications for DES. In particular, distinguishers allow identifying different instances of the same event. Moreover, [43] provides reusable components for the supervisory control of modular production systems and [40] suggests modeling guidelines for large-scale systems.

In view of the previous discussion, this paper is concerned with the formal definition, analysis and application of a general model for an AMS property that has not been investigated in the existing literature. In particular, we consider the practical scenario, where multiple products that are processed sequentially can be present in a MC. This is for example the case for a conveyor belt that can hold multiple products. If all the products have the same type, such system can be modeled as a simple buffer since all products will be processed in the same way and follow the same path after leaving the MC. However, for advanced AMSs such as FMSs or RMSs, it is possible that products with different types can enter such MC. That is, different manufacturing processes might be applied to the product on the MC or on the subsequent path through the AMS. Therefore, it is necessary to keep track of the different product types entering and leaving such MC. To the best of our knowledge, the described scenario is not considered in the literature. On the one hand, there is existing work that incorporates the product order in the system model. Nevertheless, these works either use example systems where different product types follow independent paths through the AMS [11], [16], [44] or they define the product paths such that it is not necessary to remember the product order for MCs that can hold more than one product [9], [14], [15], [34], [35], [45], [46]. On the other hand, [4], [10], [23], [25] do not take into account the product order and implicitly assume that each MC knows which product is currently transported. Practically, this assumption requires a sensor that detects the product type at each MC.

In order to address the described issue, we first discuss scenarios where the model of a MC needs to keep track of the product type and the order of products entering and leaving. As the first contribution of the paper, we define order-preserving (OP) languages over an alphabet with input events and corresponding output events based on this discussion. As the distinctive property, the order of input and corresponding output events in any string of an OP language is the same. We further introduce the notion of capacity for OP languages. The capacity denotes the maximum difference of input and output events in any string of an OP language. Using OP languages for modeling MCs of an AMS, the capacity represents the maximum number of products that can be present in a MC. As the second contribution of the paper, we show that there is a supremal OP language that is recognized by a finite state automaton, whose state count depends on the capacity and number of input events. As the main contribution of the paper, we prove that composing two supremal OP languages and projecting the resulting language to the input and output events of the first and second language, respectively, again leads to a supremal OP language. The capacity of this OP language is the sum of the capacities of the first and second language and the projection turns out to be a natural observer. Hence, it is possible to obtain an efficient representation of AMSs with OP MCs that is suitable for nonblocking supervisory control of large-scale AMSs [14], [15], [34]. Since the OP model is general, it can be used together with non-OP models for AMS, whose MCs have different properties. We demonstrate the practicability of the proposed modeling method by an FMS example with multiple products, OP and non-OP MCs and overlapping product paths. In this paper, we use a particular model of MCs with input events and output events that characterize the arrival and departure of products, respectively. We note that such model is also used in existing work such as [47] but in a different context. We further remark that automata models that preserve the order of certain events are also used in the recent papers [48], [49]. Different from this paper, our previous work in [48] focuses on the the algorithmic construction of models for MCs including MCs that preserve the order but without a theoretical analysis. Likewise, the work in [49] uses automata models that preserve the order of message transmissions and receptions for FIFO channels in networked DES without a comprehensive analysis and without considering the composition of such models.

The remainder of the paper is organized as follows. Section II summarizes the necessary background information on the supervisory control of DES and provides a problem statement based on a motivating example. Section III introduces the notion of OP languages, analyzes their most important properties and discusses the usage of OP languages for supervisory control. An FMS example in Section IV demonstrates the applicability of the defined OP models and illustrates the advantages when applying the composition of OP languages. Conclusions and ideas for future work are presented in Section V.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

II. PRELIMINARIES

This section provides the necessary notation for the formal developments in the paper. Section II-A introduces the notions related to discrete event systems (DES) and Section II-B gives background information on the supervisory control theory. In addition, Section II-C states the problem considered in this paper based on a motivating example.

A. DISCRETE EVENT SYSTEMS (DES)

We characterize the behavior of a DES by formal languages over finite alphabets Σ. Each element $\sigma \in \Sigma$ is denoted as an event, $\Sigma^\star$ is the set of all finite strings over Σ and $s_1\, s_2 \in \Sigma^\star$ defines the concatenation of two strings $s_1, s_2 \in \Sigma^\star$. $s_1 \leq s$ indicates that s_1 is a prefix of s and $|s|$ is the length of s, that is, the number of events in s. $\epsilon \in \Sigma^\star$ is the empty string with $|\epsilon| = 0$ and it holds that $s\epsilon = \epsilon s = s$ for all $s \in \Sigma^\star$. For any string $s = \sigma_1, \ldots, \sigma_{|s|} \in \Sigma^\star$ with $\sigma_i \in \Sigma$ for $i = 1, \ldots, |s|$, we further write $\mathrm{pre}_k(s) = \sigma_1 \ldots \sigma_k$ for the prefix of s with the first k events and $\mathrm{suf}_k(s) = \sigma_{k+1} \cdots \sigma_{|s|}$ for the suffix of s after the first k events. In particular, $s = \mathrm{pre}_k(s)\mathrm{suf}_k(s)$ for any $k \leq |s|$, $\mathrm{pre}_0(s) = \epsilon$ and $\mathrm{suf}_0(s) = s$ for any string $s \in \Sigma^\star$.

A formal language over Σ is a subset $L \subseteq \Sigma^\star$. $\overline{L} := \{s_1 \in \Sigma^\star \,|\, \exists s \in L \text{ s.t. } s_1 \leq s\}$ defines the prefix closure of L, and L is called prefix closed if $L = \overline{L}$.

Consider two alphabets Σ_1, Σ_2 and their union $\Sigma = \Sigma_1 \cup \Sigma_2$. Then, the natural projection $p_i : \Sigma^\star \to \Sigma_i^\star$, $i = 1, 2$, is defined iteratively such that (1) $p_i(\epsilon) := \epsilon$; (2) for $s \in \Sigma^\star$, $\sigma \in \Sigma$: $p_i(s\sigma) := p_i(s)\sigma$ if $\sigma \in \Sigma_i$, or $p_i(s\sigma) := p_i(s)$ otherwise. The set-valued inverse of p_i is written as $p_i^{-1} : \Sigma_i^\star \to 2^{\Sigma^\star}$, $p_i^{-1}(t) := \{s \in \Sigma^\star \,|\, p_i(s) = t\}$. Using the natural projection, the synchronous composition $L_1||L_2 \subseteq \Sigma^\star$ of two languages $L_i \subseteq \Sigma_i^\star$ is computed as $L_1||L_2 = p_1^{-1}(L_1) \cap p_2^{-1}(L_2) \subseteq \Sigma^\star$. In addition, we say that L_1 and L_2 are non-conflicting if it holds that

$$\overline{L_1||L_2} = \overline{L}_1||\overline{L}_2. \tag{1}$$

We model a DES as a finite state automaton $G = (X, \Sigma, \delta, x_0, X_\mathrm{m})$, with the finite set of states X; the finite alphabet of events Σ; the partial transition function $\delta : X \times \Sigma \to X$; the initial state $x_0 \in X$ and the set of marked states $X_\mathrm{m} \subseteq X$. Hereby, $\delta(x, \sigma)!$ Ã■s written if δ is defined at (x, σ) and we extend the transition function δ to a partial function on $X \times \Sigma^\star$ in the usual way. The behavior of G is given by its closed language $L(G) := \{s \in \Sigma^\star | \delta(x_0, s)!\}$ and its marked language $L_\mathrm{m}(G) := \{s \in L(G) | \delta(x_0, s) \in X_\mathrm{m}\}$. We say that G is nonblocking if $L(G) = \overline{L_\mathrm{m}(G)}$. The synchronous composition $G_1||G_2$ of two automata G_1, G_2 is defined in the usual way [22], whereby it holds that $L(G_1||G_1) = L(G_1)||L(G_2)$ and $L_\mathrm{m}(G_1||G_2) = L_\mathrm{m}(G_1)||L_\mathrm{m}(G_2)$.

B. SUPERVISORY CONTROL

In supervisory control, we write $\Sigma = \Sigma_\mathrm{c} \dot\cup \Sigma_\mathrm{u}$ for controllable (Σ_c) and uncontrollable (Σ_u) events. We say that an automaton $S = (Q, \Sigma, \nu, q_0, Q_\mathrm{m})$ is a supervisor for a plant G if S can only disable events in Σ_c. In particular, it must hold for all $s \in L(G) \cap L(S)$ and $\sigma \in \Sigma_\mathrm{u}$ with $s\sigma \in L(G)$ that also $s\sigma \in L(S)$. Then, $L(G)||L(S)$ and $L_\mathrm{m}(G)||L_\mathrm{m}(S)$ represent the closed and marked behavior of the closed-loop system $G||S$, respectively.

A language $K \subseteq L_\mathrm{m}(G)$ is said to be controllable for $L(G)$ and Σ_u if $\overline{K}\Sigma_\mathrm{u} \cap L(G) \subseteq \overline{K}$. There exists a supervisor S such that $L_\mathrm{m}(G||S) = K$ if and only if K is controllable for $L(G)$ and Σ_u [21]. In case, K is not controllable for $L(G)$ and Σ_u, the supervisor will implement the supremal controllable sublanguage of K. We write $L_m(S||G) = \mathrm{SupC}(K, G, \Sigma_\mathrm{u})$. It is ensured that such supervisor is nonblocking and maximally permissive if $\mathrm{SupC}(K, G, \Sigma_\mathrm{u}) \neq \emptyset$ [50].

C. PROBLEM STATEMENT

The work in this paper is motivated by an observation from modeling complex AMSs, where (i) different product types can be processed on the same MC and (ii) MCs might have different capacities in the sense of being able to hold several products that are then processed sequentially. In order to illustrate this observation, we consider different conveyor belts (CBs) as shown in Fig. 1.

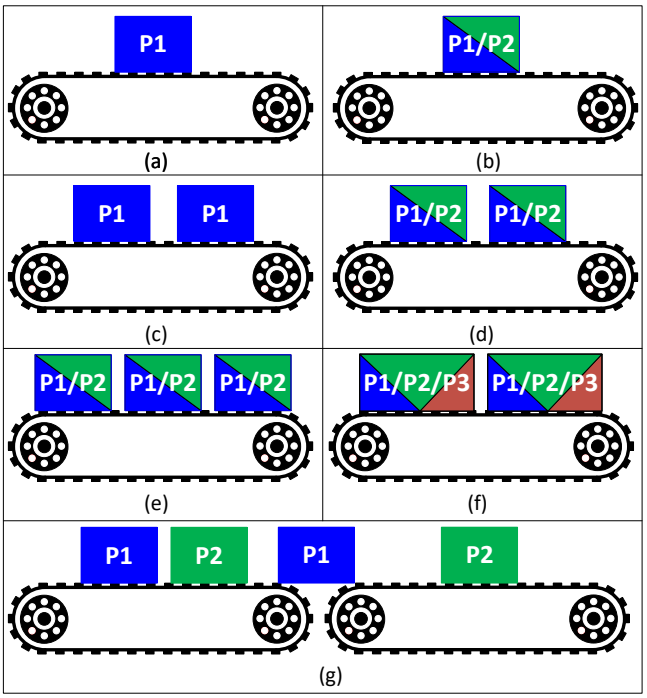

FIGURE 1: CBs with different product types and capacities: (a) one product type and capacity one; (b) two product types and capacity one; (c) one product type and capacity two; (d) two product types and capacity two; (e) two product types and capacity three; (f) three product types and capacity two; (g) two connected CBs with different product types.

We next discuss the cases in the different parts of Fig. 1 together with possible models. Hereby, we introduce events such as $\mathtt{in}_{\mathrm{P1}}$, $\mathtt{in}_{\mathrm{P2}}$, ... for products entering the CB and $\mathtt{out}_{\mathrm{P1}}$, $\mathtt{out}_{\mathrm{P2}}$, ... for products leaving the CB. Part (a) and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

(b) show the case of a CB with one and two product types and a capacity of one product. In addition, part (c) considers the case of one product type and a capacity of two. These cases can be straightforwardly modeled by the automata in Fig. 2 (a), (b) and (c). In particular, it can be directly seen that the order of product types entering and leaving the CB is either irrelevant or is preserved by these models. Differently, the modeling becomes more involved in part (d) of Fig. 1. Here, the CB has a capacity of two products and there are two different product types. That is, products of different types will leave the CB in the same order as entering the CB. In this case, the model in Fig. 2 (d-1), which is commonly used to model the scenario with two product types is not suitable. Specifically, such model suggests that product type P2 can leave the CB before type P1 even P1 enters the CB first (string $\mathtt{in_{P1}\,in_{P2}\,out_{P2}}$). A model that captures the correct order of product types entering and leaving the CB is shown in Fig. 2 (d-2). Here, only $\mathtt{out_{P1}}$ is possible in state 5 (after $\mathtt{in_{P1}in_{P2}}$) and only $\mathtt{out_{P2}}$ is possible in state 6 (after $\mathtt{in_{P2}in_{P1}}$). More complicated scenarios can easily be envisaged by increasing the capacity (as in Fig. 1 (e)) and/or the number of product types (as in Fig. 1 (f)). Moreover, it is possible to consider the connection of multiple CBs (or other MCs) that keep the order of product types as in Fig. 1 (g).

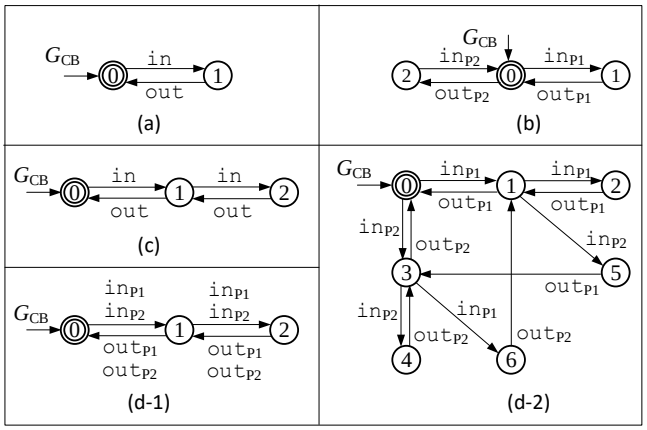

FIGURE 2: CB models for different cases: (a) one product type and capacity one; (b) two product types and capacity one; (c) one product type and capacity two; (d-1) two product types and capacity two with arbitrary order; (d-2) two product types, capacity two and preserving the product order.

In view of the previous discussion, the main aim of this paper is the construction of order-preserving (OP) models for AMSs that keep the order of product types entering and leaving each MC. Instead of constructing a separate model for each scenario, we develop a general OP DES model that is parametrized by the possible product types and the capacity of the MC. In addition, we show that reduced models can be constructed when composing such OP models from multiple MCs.

It is interesting to note that, although the existing literature considers the control of AMSs with different product types [9]–[11], [14]–[16], [23], [25], [34], [35], [44]–[46], none of the existing works develops DES models that preserve the order of products. In order to avoid the need for OP DES models, existing work restricts the general case by means of the control specification. In particular, two special cases can be observed.

The first special case is restricted to AMSs, where different product types use disjoint paths through the system and are hence processed by different MCs [11], [16], [44]. In this case, products need not be distinguished and models as in Fig. 2 (a) and (c) are suitable. In the second case, it is possible that the paths of different products intersect [9], [10], [14], [15], [23], [25], [34], [35], [45], [46]. Although this implies that the same MC is passed by multiple products, these research works again make simplifying assumptions. One the one hand, [9], [14], [15], [34], [35], [45], [46] choose the control specification such that remembering the product order is straightforward. Specifically, this is achieved by using MCs, whose capacity does not exceed one. On the other hand, [10], [23], [25] allow for MCs with multiple product types and a capacity that is greater than one. However, these papers use models as in Fig. 2 (d-1) for such MCs. That is, the order of products entering and leaving the MC is not taken into account. This means that knowledge about which product leaves such MC must be implicitly known. In a real application this implies that sensors need to be installed to detect the product type leaving such MC. In addition, using a model as in Fig. 2 (d-1) for an OP MC means that the model contains unnecessary behavior.

Accordingly, this paper focuses on the problem of developing a formal framework for OP DES models of MCs with multiple product types and capacities that are greater than one. Although an initial discussion of OP DES models is given in [48], that paper is restricted to the usage of such model for example applications. Differently, this paper provides a comprehensive formal analysis including the construction of supremal OP DES models and the composition of OP DES models.

III. ORDER-PRESERVING LANGUAGES

This section defines and analyzes OP DES models as discussed in the previous section. First, Section III-A introduces the required notation for the definition of OP languages and determines their basic properties. Then, Section III-B proves the existence of a supremal OP language and Section III-C studies the composition of OP languages. Finally, Section III-D shows that reduced OP models that are suitable for supervisory control can be obtained after composition.

A. NOTATION AND DEFINITIONS

We consider an alphabet Σ that is divided into disjoint sets of input events Σ^{in} and output events Σ^{out} such that $\Sigma^{\text{in}} \cup \Sigma^{\text{out}} = \Sigma$ and $\Sigma^{\text{in}} \cap \Sigma^{\text{out}} = \emptyset$. Moreover, we introduce the natural projections to the respective alphabets as $p^{\text{in}} : \Sigma^\star \to (\Sigma^{\text{in}})^\star$ and $p^{\text{out}} : \Sigma^\star \to (\Sigma^{\text{out}})^\star$. In addition, we define a bijective input/output mapping $m : \Sigma^{\text{in}} \to \Sigma^{\text{out}}$ such that for each input event $\alpha \in \Sigma^{\text{in}}$, $m(\alpha) \in \Sigma^{\text{out}}$ denotes the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

corresponding output event. We further write $m^{-1} : \Sigma^{\text{out}} \to \Sigma^{\text{in}}$ for the inverse mapping.

Relating this definition to the discussion about AMSs in the previous section, an input event $\alpha \in \Sigma^{\text{in}}$ and output event $m(\alpha)$ represent a product entering and leaving a MC, respectively.

Given the above notation, we introduce the new notion of an OP language, whose strings preserve the order of input events and their corresponding output events.

Definition 1. *Consider Σ, Σ^{in}, Σ^{out}, p^{in}, p^{out} and m as introduced above. A string $s \in \Sigma^\star$ is denoted as OP if*

$$m(p^{\text{in}}(s)) = p^{\text{out}}(s). \tag{2}$$

A language $L \subseteq \Sigma^\star$ is denoted as OP if it holds for all $s \in L$ that s is OP.

That is, each OP string has the same order of input events and corresponding output events. In particular, it must be the case that each OP string $s \in \Sigma^\star$ has the same number of input and output events: $|p^{\text{in}}(s)| = |p^{\text{out}}(s)|$ and each prefix $s' \leq s$ of an OP string s must fulfill

$$p^{\text{out}}(s') \leq m(p^{\text{in}}(s')).$$

This means that an output event can only occur after its corresponding input event.

For illustration, we consider the languages $L_1 = L_{\text{m}}(G_1)$ and $L_2 = L_{\text{m}}(G_2)$ of the automata G_1 and G_2 in Fig. 3. Here, we assume that $\Sigma = \{\mathtt{a}_1, \mathtt{a}_2, \mathtt{b}_1, \mathtt{b}_2\}$, $\Sigma^{\text{in}} = \{\mathtt{a}_1, \mathtt{a}_2\}$, $\Sigma^{\text{out}} = \{\mathtt{b}_1, \mathtt{b}_2\}$ and $m(\mathtt{a}_1) = \mathtt{b}_1$, $m(\mathtt{a}_2) = \mathtt{b}_2$. For example, it holds that $s_1 = \mathtt{a}_1\mathtt{a}_2\mathtt{b}_1\mathtt{b}_2 \in L_1$ and $s_2 = \mathtt{a}_2\mathtt{b}_2\mathtt{a}_1\mathtt{a}_2\mathtt{b}_1\mathtt{b}_2 \in L_2$ are OP strings. Specifically, $m(p^{\text{in}}(s_1)) = m(\mathtt{a}_1\mathtt{a}_2) = \mathtt{b}_1\mathtt{b}_2 = p^{\text{out}}(s_1)$ and $m(p^{\text{in}}(s_2)) = m(\mathtt{a}_2\mathtt{a}_1\mathtt{a}_2) = \mathtt{b}_2\mathtt{b}_1\mathtt{b}_2 = p^{\text{out}}(s_2)$. It can be further verified that L_2 is an OP language. Nevertheless, it is the case that L_1 is not OP. This can be seen by looking at the string $s_3 = \mathtt{a}_1\mathtt{a}_2\mathtt{b}_2\mathtt{b}_1 \in L_1$ with $m(p^{\text{in}}(s_3)) = \mathtt{b}_1\mathtt{b}_2 \neq p^{\text{out}}(s_3) = \mathtt{b}_2\mathtt{b}_1$.

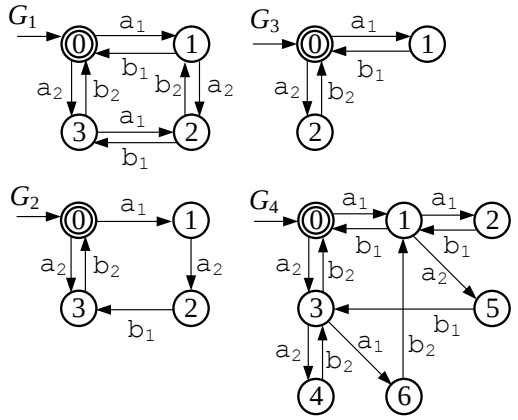

FIGURE 3: Example automata for OP languages.

Referring to Definition 1, we write

$$\mathcal{L}^{\text{OP}}(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m) = \{L \subseteq \Sigma^\star | L \text{ is OP}\} \tag{3}$$

for the set of all OP languages with alphabet $\Sigma = \Sigma^{\text{in}} \cup \Sigma^{\text{out}}$ and the map m. In addition, we introduce the notion of the capacity of an OP language.

Definition 2. *Assume Σ^{in}, Σ^{out}, Σ, p^{in} and p^{out} are given as above. Consider an OP string $s \in \Sigma^\star$. Then, the capacity of s is defined by the function $c : \Sigma^\star \to \mathbb{N}$ as*

$$c(s) = \max_{s' \leq s} \{|p^{\text{in}}(s')| - |p^{\text{out}}(s')\}. \tag{4}$$

Extending this notion to languages, the capacity of any OP language $L \in \mathcal{L}^{\text{OP}}(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m)$ is written as

$$c(L) = \max_{s \in L} \{c(s)\}. \tag{5}$$

That is, the capacity of an OP string $s \in \Sigma^\star$ captures the maximum difference between the number of input events and the number of output events in any prefix s' of s. Analogously, the capacity of an OP language is given by the maximum capacity of any string $s \in L$.

Consider the OP languages $L_2 = L_{\text{m}}(G_2)$ and $L_3 = L_{\text{m}}(G_3)$ of the automata G_2 and G_3 in Fig. 3. It holds that $c(L_2) = 2$ since the largest difference between the number of input and output events is obtained for strings $s \in L_2$ that contain the substring $\mathtt{a}_1\mathtt{a}_2$. Differently, $c(L_3) = 1$ since each input event $\mathtt{a}_i$, $i = 1, 2$, is directly followed by the corresponding output event $\mathtt{b}_i$.

In order to relate the notion of capacity to practical systems, we consider AMSs as discussed in the previous section. Here, input events could be identified with products of certain types entering a MC, whereas output events can characterize the corresponding products exiting a MC. Therefore, modeling an AMS by an OP language L ensures that products leave the system in the order of entering the system. In that case, the capacity $c(L)$ indicates the maximum number of products that can be in the system simultaneously.

B. SUPREMAL ORDER-PRESERVING LANGUAGE

In order to analyze properties of OP languages, we introduce

$$\mathcal{L}^{\text{OP}}_C(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m) = \{L \in \mathcal{L}^{\text{OP}}(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m) | c(L) = C\} \tag{6}$$

as the set of OP languages with alphabet $\Sigma = \Sigma^{\text{in}} \cup \Sigma^{\text{out}}$ and capacity C. Our first result shows that OP languages of a given capacity C are closed under arbitrary union with a supremal element $L^{\text{sup}}_C(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m)$.

Theorem 1. *Let $\mathcal{L}^{\text{OP}}_C(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m)$ be as defined in (6). Then, $\mathcal{L}^{\text{OP}}_C(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m)$ is closed under arbitrary union and contains a supremal element*

$$L^{\text{sup}}_C(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m) = \bigcup_{L \in \mathcal{L}^{\text{OP}}_C(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m)} L. \tag{7}$$

We note that, for ease of notation, we write $\mathcal{L}^{\text{OP}}$, $\mathcal{L}^{\text{OP}}_C$ and L^{sup}_C whenever Σ^{in}, Σ^{out} and m are clear from the context.

Proof. We first show that $\mathcal{L}^{\text{OP}}_C$ is closed under arbitrary union. To this end, let $L_1, L_2 \in \mathcal{L}^{\text{OP}}_C$. We have to show that $L_1 \cup L_2 \in \mathcal{L}^{\text{OP}}_C$. Take an arbitrary string $s \in L_1 \cup L_2$. Then,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

$s \in L_1$ or $s \in L_2$. In both cases, it holds that s is OP since L_1 and L_2 are both OP. Hence, it follows that $L_1 \cup L_2$ is OP. Moreover, $c(s) \leq C$ since $c(L_1) = c(L_2) = C$. It remains to show that there exists at least one string $s' \in L_1 \cup L_2$ such that $c(s') = C$. Without loss of generality, we can take some $s' \in L_1 \subseteq L_1 \cup L_2$ such that $c(s') = C$ since $c(L_1) = C$. Together, we conclude that $L_1 \cup L_2$ is OP and $c(L_1 \cup L_2) = C$. Hence, $L_1 \cup L_2 \in \mathcal{L}_C^{\mathrm{OP}}$. Now, consider $L_C^{\sup} = \bigcup_{L \in \mathcal{L}_C^{\mathrm{OP}}} L$. Since $\mathcal{L}_C^{\mathrm{OP}}$ is closed under arbitrary union, it must hold that $L_C^{\sup} \in \mathcal{L}_C^{\mathrm{OP}}$. Hence, $L_C^{\sup}$ is indeed the supremal OP language with capacity C. □

In order to further characterize $L_C^{\sup}$, we first define the function

$$f : \overline{L}_C^{\sup} \to (\Sigma^{\mathrm{in}})^\star : f(s) = \mathrm{suf}_{|p^{\mathrm{out}}(s)|}(p^{\mathrm{in}}(s)) \quad (8)$$

for any given $s \in \overline{L}_C^{\sup}$. That is, $f(s)$ represents the string of input events in s, whose corresponding output events did not occur, yet. Consider for example the string $s = \mathtt{a}_1\mathtt{a}_2\mathtt{b}_1\mathtt{b}_2\mathtt{a}_2\mathtt{b}_2\mathtt{a}_1\mathtt{a}_2 \in \overline{L}_2 = L(G_2)$ in Fig. 3 with $f(s) = \mathrm{suf}_{|\mathtt{b}_1\mathtt{b}_2\mathtt{b}_2|}(p^{\mathrm{in}}(s)) = \mathrm{suf}_3(\mathtt{a}_1\mathtt{a}_2\mathtt{a}_2\mathtt{a}_1\mathtt{a}_2) = \mathtt{a}_1\mathtt{a}_2$. Then, we introduce the equivalence relation $\equiv_{\mathrm{OP}}$ as the equivalence kernel of f such that for any two strings $s, s' \in \overline{L}_C^{\sup}$,

$$s \equiv_{\mathrm{OP}} s' \iff f(s) = f(s'). \quad (9)$$

Using $\equiv_{\mathrm{OP}}$, we construct the automaton $G_C^{\sup} = (X_C^{\sup}, \Sigma, \delta_C^{\sup}, x_{0,C}^{\sup}, X_{\mathrm{m},C}^{\sup})$ as a recognizer of $L_C^{\sup}$. The state set of $G_C^{\sup}$ is given by the quotient set $X_C^{\sup} = \overline{L}_C^{\sup} / \equiv_{\mathrm{OP}}$ of $\overline{L}_C^{\sup}$. Since any string $s \in L_C^{\sup}$ has a capacity $c(s) \leq C$, it must be the case that $|f(s)| = |p^{\mathrm{in}}(s)| - |p^{\mathrm{out}}(s)| \leq C$ for any $s \in \overline{L}_C^{\sup}$. Hence, the quotient set

$$X_C^{\sup} = \{[\epsilon]\} \cup \{[\alpha_1, \ldots, \alpha_j] | j \in \{1, \ldots, C\} \wedge \alpha_i \in \Sigma^{\mathrm{in}}, \forall i = 1, \ldots, j\}$$

consists of the equivalence class $[\epsilon]$ and one equivalence class $[\alpha_1 \cdots \alpha_j]$ for each possible sequence $\alpha_1 \cdots \alpha_j$ of j arbitrary input events $\alpha_i \in \Sigma^{\mathrm{in}}$, $1 \leq i \leq j$, whereby $1 \leq j \leq C$. The initial state $x_{0,C}^{\sup} = [\epsilon]$ corresponds to $s = \epsilon \in \overline{L}_C^{\sup}$ and the set of marked states is $X_{\mathrm{m},C}^{\sup} = \{[\epsilon]\}$. Furthermore, the transition relation of $G_C^{\sup}$ is defined with the following rules:

- R1 $\delta_C^{\sup}([\epsilon], \alpha_1) = [\alpha_1]$ for all $\alpha_1 \in \Sigma^{\mathrm{in}}$,
- R2 $\delta_C^{\sup}([\alpha_1], \omega_1) = [\epsilon]$ for $\omega_1 = m(\alpha_1)$,
- R3 $\delta_C^{\sup}([\alpha_1, \ldots, \alpha_j], \alpha_{j+1}) = [\alpha_1, \ldots, \alpha_j \alpha_{j+1}]$ for all $[\alpha_1, \ldots, \alpha_j] \in X_C^{\sup}$ and $\alpha_{j+1} \in \Sigma^{\mathrm{in}}$, $j = 1, \ldots, C-1$,
- R4 $\delta_C^{\sup}([\alpha_1, \alpha_2, \ldots, \alpha_j], \omega_1) = [\alpha_2, \ldots, \alpha_j]$ for $\omega_1 = m(\alpha_1)$ and for all $[\alpha_1, \alpha_2, \ldots, \alpha_j] \in X_C^{\sup}$, $j = 2, \ldots, C$.

That is, $\delta_C^{\sup}$ is defined such that the state of $G_C^{\sup}$ keeps track of the sequence of input events, whose corresponding output events did not occur, yet. Specifically, R1 and R3 address the case where an input event occurs. This input event is hence appended to the current sequence of input events. Conversely, the first input event α_1 is removed from the sequence of input events in R2 and R4 if its corresponding output event $\omega_1 = m(\alpha_1)$ occurs. Here, it is important to note that only the output event of the first input event in the sequence can occur so as to preserve the order of input and output events. In addition, the marked state $[\epsilon]$ corresponds to the case where the sequence of input events, whose corresponding output events did not occur, yet, is empty. That is, strings $s \in L_{\mathrm{m}}(G_C^{\sup})$ have the property that $m(p^{\mathrm{in}}(s)) = p^{\mathrm{out}}(s)$ and hence belong to $L_C^{\sup}$.

We next state Lemma 1, which formalizes the relationship between $L_C^{\sup}$ and $G_C^{\sup}$.

Lemma 1. *Consider $L_C^{\sup}$ as defined in (7) and $G_C^{\sup}$ as introduced above. It holds that*

(i) $s \in L_{\mathrm{m}}(G_C^{\sup}) \Rightarrow s$ *is OP and* $c(s) \leq C$,
(ii) $s \in L_C^{\sup} \Rightarrow s \in L_{\mathrm{m}}(G_C^{\sup})$.

The detailed proof of Lemma 1 is provided in Appendix A. The lemma indicates that (i) any string in the marked language of $G_C^{\sup}$ is OP and its capacity is bounded by C and (ii) any OP string, whose capacity is bounded by C belongs to the marked language of $G_C^{\sup}$. Using Lemma 1, Theorem 2 shows that $G_C^{\sup}$ has a finite number of states and $L_{\mathrm{m}}(G_C^{\sup}) = L_C^{\sup}$.

Theorem 2. *Consider $L_C^{\sup}$ as defined in (7) and $G_C^{\sup}$ as introduced above. Then, it holds that $G_C^{\sup}$ is a canonical recognizer of $L_C^{\sup}$ and*

$$|X_C^{\sup}| = \sum_{i=0}^{C} |\Sigma^{\mathrm{in}}|^i. \quad (10)$$

In particular, $L_C^{\sup}$ is a regular language.

Proof. In order to prove that $G_C^{\sup}$ is a canonical recognizer of $L_C^{\sup}$, we first show that $L_{\mathrm{m}}(G_C^{\sup}) = L_C^{\sup}$. That is, $L_{\mathrm{m}}(G_C^{\sup}) \subseteq L_C^{\sup}$ and $L_C^{\sup} \subseteq L_{\mathrm{m}}(G_C^{\sup})$. For the first inclusion, let $s \in L_{\mathrm{m}}(G_C^{\sup})$. Then, it follows from Lemma 1 (i) that s is OP and $c(s) \leq C$. That is, $s \in L_C^{\sup}$. The second inclusion is directly implied by (ii) in Lemma 1.

In addition, consider an arbitrary state $x = [\alpha_1 \cdots \alpha_j] \in X_C^{\sup}$ for $1 \leq j \leq C$ and $\alpha_i \in \Sigma^{\mathrm{in}}$ for $1 \leq i \leq j$. Applying R4 iteratively, it holds that $\delta_C^{\sup}(x, \omega_1 \cdots \omega_{j-1}) = [\alpha_j] \in X_C^{\sup}$ with $\omega_i = m(\alpha_i)$, $i = 1, \ldots, j-1$. Furthermore, $\delta_C^{\sup}([\alpha_j], \omega_j) = [\epsilon] \in X_{\mathrm{m},C}^{\sup}$ for $\omega_j = m(\alpha_j)$ and with R2. Together, we have that $\delta_C^{\sup}(x, \omega_1 \cdots \omega_j) = [\epsilon] \in X_{\mathrm{m},C}^{\sup}$. Nevertheless, for any state $\hat{x} \in X_C^{\sup}$ such that $\hat{x} \neq x$, it must be the case that $\neg\delta_C^{\sup}(\hat{x}, \omega_1 \cdots \omega_j)!$ or $\delta_C^{\sup}(\hat{x}, \omega_1 \cdots \omega_j) \neq [\epsilon]$ (otherwise $\hat{x} = x$ when applying R4 and R2). Hence, none of the states in $X_C^{\sup}$ can be equivalent according to the Nerode equivalence [22]. Since $L_{\mathrm{m}}(G_C^{\sup}) = L_C^{\sup}$, this implies that $G_C^{\sup}$ is a canonical recognizer of $L_C^{\sup}$.

Finally, we consider the state set $X_C^{\sup}$. There are $|\Sigma^{\mathrm{in}}|^j$ combinations for each equivalence class $[\alpha_1 \cdots \alpha_j]$, $j = 1, \ldots, C$ and there is one equivalence class $[\epsilon]$. Hence, it follows that $G_C^{\sup}$ has a finite number of

$$|X_C^{\sup}| = \sum_{j=0}^{C} |\Sigma^{\mathrm{in}}|^j$$

states. This directly implies that $L_C^{\sup}$ is a regular language. □

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

According to Theorem 2, $L_C^{\sup}$ can be realized by a canonical recognizer $G_C^{\sup}$ with a finite number of states as given in (10) that depends on the number of input events $|\Sigma^{\text{in}}|$ and the capacity C. Hereby, each state of $G_C^{\sup}$ represents an equivalence class of $\equiv_{\text{OP}}$ as defined in (9) and the transition relation of $G_C^{\sup}$ is defined by the rules (R1) to (R4). It is hence straightforward to compute $G_C^{\sup}$. As an example, consider $\Sigma = \Sigma^{\text{in}} \cup \Sigma^{\text{out}} = \{\mathtt{a}_1, \mathtt{a}_2\} \cup \{\mathtt{b}_1, \mathtt{b}_2\}$ with $m(\mathtt{a}_1) = \mathtt{b}_1$ and $m(\mathtt{a}_2) = \mathtt{b}_2$. Then, Fig. 3 shows $G_3 = G_1^{\sup}$ and $G_4 = G_2^{\sup}$.

Moreover, as a more practical example, we consider a CB in an AMS that can transport 3 different product types P1, P2 and P3 and that can hold up to 2 products simultaneously. It is further the case that products leave the CB in the same sequence as entering the CB. Hence, the behavior of such CB should be modeled by a supremal OP language with the input events $\Sigma^{\text{in}} = \{\mathtt{i}_{\text{P1}}, \mathtt{i}_{\text{P2}}, \mathtt{i}_{\text{P3}}\}$ and the output events $\Sigma^{\text{out}} = \{\mathtt{o}_{\text{P1}}.\mathtt{o}_{\text{P2}}, \mathtt{o}_{\text{P3}}\}$. The mapping m is defined by $m(\mathtt{i}_{\text{Pi}}) = \mathtt{o}_{\text{Pi}}$ for $\mathtt{i} = 1, 2, 3$ and the capacity is $C = 2$. According to (10), the canonical recognizer G_{CB} for the CB as shown in Fig. 4 has 13 states. Hereby, there is one product on the CB in the states shaded in light gray, whereas there are two products on the CB in the states shaded in dark gray.

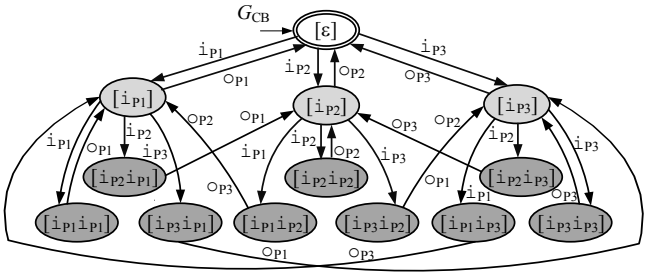

FIGURE 4: OP DES model of a CB with three product types P1, P2, P3 and capacity $C = 2$.

C. COMPOSITION OF ORDER-PRESERVING LANGUAGES

Practical AMSs are commonly composed of various modular system components. That is, we next investigate the composition of multiple MCs with OP languages. To this end, we first consider the case of two OP languages $L_i \subseteq \Sigma_i^\star$ over the alphabet Σ_i with given capacities $c(L_i) = C_i$ and maps $m_i : \Sigma_i^{\text{in}} \to \Sigma_i^{\text{out}}$ for $i = 1, 2$. In addition, we assume that the output events of the first language are the input events of the second language: $\Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$. In an AMS, such languages could represent two MCs, whereby the first MC delivers products to the second MC. Defining the alphabets $\Sigma^{\text{in}} = \Sigma_1^{\text{in}}$, $\Sigma^{\text{out}} = \Sigma_2^{\text{out}}$, $\Sigma = \Sigma^{\text{in}} \cup \Sigma^{\text{out}}$, the natural projections $p : (\Sigma_1 \cup \Sigma_2)^\star \to \Sigma^\star$, $p_i : (\Sigma_1 \cup \Sigma_2)^\star \to \Sigma_i^\star$, $i = 1, 2$ and the mapping $m : \Sigma^{\text{in}} \to \Sigma^{\text{out}} : m(s) = m_2(m_1(s))$, we determine the composed language

$$L = p(L_1||L_2). \tag{11}$$

That is, we project $L_1||L_2$ to the alphabet Σ, which only contains the input and output events of the composed language. We next show that L is an OP language with capacity $C \leq C_1 + C_2$.

Theorem 3. *Assume that* Σ_1^{in}, Σ_1^{out}, Σ_2^{in}, Σ_2^{out}, m_1, m_2, Σ, m *and* p *are as introduced above with* $\Sigma^{in} = \Sigma_1^{in}$, $\Sigma^{out} = \Sigma_2^{out}$ *and* $\Sigma_1^{out} = \Sigma_2^{in}$. *Let* $L_1 \in \mathcal{L}_{C_1}^{\text{OP}}(\Sigma_1^{\text{in}}, \Sigma_1^{\text{out}}, m_1)$, $L_2 \in \mathcal{L}_{C_2}^{\text{OP}}(\Sigma_2^{\text{in}}, \Sigma_2^{\text{out}}, m_2)$ *and suppose that* $L_1||L_2 \neq \emptyset$. *Then*

$$L = p(L_1||L_2) \in \mathcal{L}_C^{OP}(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m) \tag{12}$$

for some $C \leq C_1 + C_2$.

Proof. Since $L_1||L_2 \neq \emptyset$, there is at least one string $s \in p(L_1||L_2)$. Now consider any arbitrary string $s \in p(L_1||L_2)$. We need to show that s is OP and $c(s) \leq C_1 + C_2$.

We first show that s is OP. Assume the contrary. That is $s \in L$ and s is not OP. Then, we can write $s = u_1\alpha u_2\omega u_3$ with $u_1, u_2, u_3 \in \Sigma^\star$, $\alpha \in \Sigma^{\text{in}}$ and $\omega \in \Sigma^{\text{out}}$, such that for some $k < |p^{\text{in}}(s)||$

$$\text{pre}_k m(p^{\text{in}}(s)) = m(p^{\text{in}}(u_1)) = \text{pre}_k p^{\text{out}}(s) = p^{\text{out}}(u_1\alpha u_2)$$

but

$$\begin{aligned}\text{pre}_{k+1} m(p^{\text{in}}(s)) &= m(p^{\text{in}}(u_1\alpha)) \neq \text{pre}_{k+1} p^{\text{out}}(s) \\ &= p^{\text{out}}(u_1\alpha u_2\omega) = p^{\text{out}}(u_1\alpha u_2)\omega.\end{aligned}$$

In words, we consider that the order of the first k input events in s is equal to the order of the first k output events, whereas the order of the first $k+1$ input events of s is different from the order of the first $k+1$ output events. Since $m(p^{\text{in}}(u_1)) = p^{\text{out}}(u_1\alpha u_2)$, this implies that $m(\alpha) \neq \omega$.

We further know that $s \in L = p(L_1||L_2)$ implies that there are $s_1 \in L_1$ and $s_2 \in L_2$ such that $s = p(s_1||s_2)$. Since L_i is OP for $i = 1, 2$, also s_i is OP for $i = 1, 2$. Considering that $\Sigma_1^{\text{in}} = \Sigma^{\text{in}}$, $\Sigma_2^{\text{out}} = \Sigma^{\text{out}}$, $\Sigma_1 \cap \Sigma_2^{\text{out}} = \emptyset$ and $\Sigma_2 \cap \Sigma_1^{\text{in}} = \emptyset$, we further conclude that $p_1^{\text{in}}(s_1) = p^{\text{in}}(s)$, $p_2^{\text{out}}(s_2) = p^{\text{out}}(s)$ and $p_1^{\text{out}}(s_1) = p_2^{\text{in}}(s_2)$.

Accordingly, we determine $\text{pre}_{k+1} m_1(p_1^{\text{in}}(s_1)) = \text{pre}_{k+1} m_1(p^{\text{in}}(s)) = m_1(p^{\text{in}}(u_1\alpha)) = m_1(p^{\text{in}}(u_1))\alpha$ and $\text{pre}_{k+1} m_2^{-1}(p_2^{\text{out}}(s_2)) = \text{pre}_{k+1} m_2^{-1}(p^{\text{out}}(s)) = m_2^{-1}(p^{\text{out}}(u_1\alpha u_2\omega)) = m_2^{-1}(p^{\text{out}}(u_1\alpha u_2))m_2^{-1}(\omega)$. Since $m(p^{\text{in}}(u_1)) = m_2(m_1(p^{\text{in}}(u_1))) = p^{\text{out}}(u_1\alpha u_2)$ and $\Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$, we conclude that $\text{pre}_k m_1(p_1^{\text{in}}(s_1)) = \text{pre}_k m_2^{-1}(p_2^{\text{out}}(s_2))$. However,

$$\begin{aligned}&\text{pre}_{k+1} m_1(p_1^{\text{in}}(s_1)) = m_1(p^{\text{in}}(u_1))m_1(\alpha) \\ \neq &\text{pre}_{k+1} m_2^{-1}(p_2^{\text{out}}(s_2)) = m_2^{-1}(p^{\text{out}}(u_1\alpha u_2))m_2^{-1}(\omega)\end{aligned}$$

because $m_1(\alpha) \neq m_2^{-1}(\omega)$ (which follows from $m(\alpha) \neq \omega$). Hence, $p_1^{\text{out}}(s_1) \neq p_2^{\text{in}}(s_2)$ such that $p_1^{-1}(s_1) \cap p_2^{-1}(s_2) = \emptyset$, which implies that $s \notin p(p_1^{-1}(s_1) \cap p_2^{-1}(s_2)) \subseteq p(L_1||L_2) = L$. This is a contradiction, because we assumed that $s \in L$.

It remains to show that $c(L) = C \leq C_1 + C_2$. To this end, let $s \in p(L_1||L_2)$ and $s_1 \in L_1$, $s_2 \in L_2$ such that $s = p(s_1||s_2)$. From the previous discussion, we know that for each $s' \leq s$, $p^{\text{in}}(s') = p_1^{\text{in}}(s_1')$, $p^{\text{out}}(s') = p_2^{\text{out}}(s_2')$ and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

$p_1^{\text{out}}(s_1') = p_2^{\text{in}}(s_2')$ for some $s_1' \leq s_1$, $s_2' \leq s_2$. That is, we compute

$$\begin{aligned} c(s) &= \max_{s' \leq s} \{|p^{\text{in}}(s')| - |p^{\text{out}}(s')|\} \\ &= \max_{s_1' \leq s_1, s_2' \leq s_2, s' = p(s_1'||s_2')} \{|p_1^{\text{in}}(s_1')| - |p_2^{\text{out}}(s_2')|\} \\ &= \max_{s_1' \leq s_1, s_2' \leq s_2, s' = p(s_1'||s_2')} \{\underbrace{|p_1^{\text{in}}(s_1')| - |p_1^{\text{out}}(s_1')|}_{\leq C_1} \\ &\quad + \underbrace{|p_2^{\text{in}}(s_2')| - |p_2^{\text{out}}(s_2')|}_{\leq C_2}\} \leq C_1 + C_2. \end{aligned}$$

Since $s \in L$ was arbitrary, it follows that $c(L) = C \leq C_1 + C_2$. That is, $L \in \mathcal{L}_C^{OP}(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m)$ for some $C \leq C_1 + C_2$. □

We demonstrate the implications of Theorem 3 using the example automata in Fig. 3 and 5.

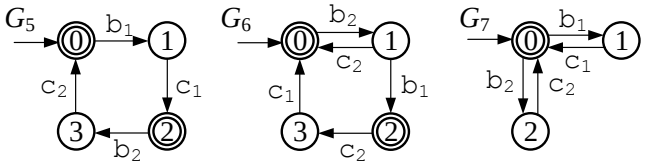

FIGURE 5: Example automata for the composition of OP languages.

First consider the languages $L_2 = L_{\text{m}}(G_2)$ with $C_2 = 2$ in Fig. 3 and $L_5 = L_{\text{m}}(G_5)$ with $C_5 = 1$ in Fig. 5. A canonical recognizer G_{25} (excluding the states shaded in gray) with $L_{25} = L_{\text{m}}(G_{25}) = p(L_2||L_5)$ and $\Sigma^{\text{in}} = \{\mathtt{a}_1, \mathtt{a}_2\}$ and $\Sigma^{\text{out}} = \{\mathtt{c}_1, \mathtt{c}_2\}$ is shown in Fig. 6. It is readily observed that L_{25} is OP and $C_{25} = c(L_{25}) = 3 = C_2 + C_5$. For illustration, the states of G_{25} are colored such that orange, blue and green states correspond to strings with capacity 1, 2 and 3, respectively. Moreover, G_{25} shows two blocking states 2 and 6 that are shaded in gray. In particular, it holds that $\overline{L_2||L_5} \neq \overline{L}_2||\overline{L}_5$ such that L_2 and L_5 are conflicting. That is, even Theorem 3 shows that any string $s \in L_{\text{m}}(G_{25})$ is OP, there might be strings in $L(G_{25})$ that cannot be extended to an OP string. Next, we consider $L_2 = L_{\text{m}}(G_2)$ with $C_2 = 2$ in Fig. 3 and $L_6 = L_{\text{m}}(G_6)$ with $C_6 = 2$ in Fig. 5. A canonical recognizer G_{26} for $L_{26} = L_{\text{m}}(G_{26}) = p(L_2||L_6)$ with $\Sigma^{\text{in}} = \{\mathtt{a}_1, \mathtt{a}_2\}$ and $\Sigma^{\text{out}} = \{\mathtt{c}_1, \mathtt{c}_2\}$ is shown in Fig. 6. Here, it turns out that $C_{26} = 3 \neq C_1 + C_2 = 4$. That is, the capacity of L_{26} is smaller than the accumulated capacity of L_2 and L_6. This is possible according to Theorem 3 and occurs since $L_2 \subset L_2^{\text{sup}}$ and $L_6 \subset L_2^{\text{sup}}$. We finally consider the composition of $L_3 = L_{\text{m}}(G_3) = L_1^{\text{sup}}$ and $L_4 = L_{\text{m}}(G_4) = L_2^{\text{sup}}$ in Fig. 3 with $L_7 = L_{\text{m}}(G_7) = L_1^{\text{sup}}$ in Fig. 5. The resulting OP languages $L_{37} = L_{\text{m}}(G_{37})$ and $L_{47} = L_{\text{m}}(G_{47})$ are shown in Fig. 6. Hereby, it is interesting to note that $L_{37} = L_2^{\text{sup}}$ and $L_{47} = L_3^{\text{sup}}$. In addition, L_3 and L_7 as well as L_4 and L_7 are non-conflicting. In the sequel, we formalize the observations from this example in Theorem 4 and 5.

Theorem 4. *Assume that* Σ_1^{in}, Σ_1^{out}, Σ_2^{in}, Σ_2^{out}, m_1, m_2, Σ, m *and* p *are as introduced above with* $\Sigma^{in} = \Sigma_1^{in}$, $\Sigma^{out} = \Sigma_2^{out}$ and $\Sigma_1^{out} = \Sigma_2^{in}$. *Further, let* $L_1 = L_{C_1}^{\text{sup}}(\Sigma_1^{\text{in}}, \Sigma_1^{\text{out}}, m_1)$ *and* $L_2 = L_{C_2}^{\text{sup}}(\Sigma_2^{\text{in}}, \Sigma_2^{\text{out}}, m_2)$ *for* $C_1, C_2 > 0$. *Then,* $L = p(L_1||L_2) = L_{C_1+C_2}^{\text{sup}}(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m)$.

Theorem 4 states that the composition of two supremal OP languages again yields a supremal OP language, whose capacity is the sum of the capacities of the composed languages.

In order to prove Theorem 4, we establish three lemmas that show general properties of $L = p(L_1||L_2)$. Lemma 2 determines that the capacity of L is $c(L) = C = C_1 + C_2$ if both languages L_i, $i = 1, 2$ are supremal OP languages with capacity C_i.

Lemma 2. *Consider the setting in Theorem 4. Then,* $c(L) = C = C_1 + C_2$.

The proof of Lemma 2 is given in Appendix B.

Lemma 3 introduces a decomposition of $f(s)$ in (8) for any string $s \in L = p(L_1||L_2)$.

Lemma 3. *Consider the setting in Theorem 4 and let* $s_1 \in \overline{L}_1$, $s_2 \in \overline{L}_2$ *and* $s \in \overline{L}_1||\overline{L}_2$ *such that* $s = p(s_1||s_2)$. *Further, let* $f : \Sigma^\star \to (\Sigma^{\text{in}})^\star$, $f_1 : \Sigma_1^\star \to (\Sigma_1^{\text{in}})^\star$ *and* $f_2 : \Sigma_2^\star \to (\Sigma_2^{\text{in}})^\star$ *be defined as in* (8). *Then, it holds that*

$$f(s) = m_1^{-1}(f_2(s_2))\, f_1(s_1). \tag{13}$$

That is, for any string $s = p(s_1||s_2) \in p(\overline{L}_1||\overline{L}_2)$ the excess input events $f(s)$, whose corresponding output events did not occur, yet, are either excess input events $f_1(s_1) \in (\Sigma_1^{\text{in}})^\star$ or correspond to excess input events $f_2(s_2) \in (\Sigma_2^{\text{in}})^\star$. The proof of Lemma 3 is given in Appendix C.

Lemma 4 shows that any extension $s\sigma \in p(\overline{L}_1||\overline{L}_2)$ of a string $s \in p(\overline{L}_1||\overline{L}_2)$ with an event $\sigma \in \Sigma$ has a corresponding extension of any string $t \in \overline{L}_1||\overline{L}_2$ that has the projection $p(t) = s$.

Lemma 4. *Consider the setting in Theorem 4. Then, it holds for any* $t \in \overline{L}_1||\overline{L}_2$ *and* $\sigma \in \Sigma$ *that*

$$\begin{aligned} p(t)\sigma \in p(\overline{L}_1||\overline{L}_2) \Rightarrow &\exists v \in (\Sigma_1 \cup \Sigma_2)^\star \text{ s.t.} \\ &p(v) = \sigma \text{ and } tv \in \overline{L}_1||\overline{L}_2. \end{aligned} \tag{14}$$

The proof of Lemma 4 is given in Appendix D.

Using Lemma 2 to 4, we next prove Theorem 4.

Proof. In order to show that $L = p(L_1||L_2) = L_{C_1+C_2}^{\text{sup}}$, we show that (i) $L \subseteq L_{C_1+C_2}^{\text{sup}}$ and (ii) $L \supseteq L_{C_1+C_2}^{\text{sup}}$.

(i) Since $L_1 = L_{C_1}^{\text{sup}}$ and $L_2 = L_{C_2}^{\text{sup}}$ for $C_1, C_2 > 0$, it holds that $\epsilon \in L_1||L_2 \neq \emptyset$. Hence, we know from Theorem 3 that L is OP and Lemma 2 implies that $c(L) = C = C_1 + C_2$. Hence, $L \subseteq L_{C_1+C_2}^{\text{sup}}$.

(ii) In order to show that $L \supseteq L_C^{\text{sup}}$, we take an arbitrary OP string $s \in L_C^{\text{sup}}$. First, we show by induction that $s \in p(\overline{L}_1||\overline{L}_2)$. To this end, we write $s = \sigma_1 \cdots \sigma_{|s|}$ with $\sigma_i \in \Sigma$ for $i = 1, \ldots, |s|$.

For the initialization, consider $s_0 = \text{pre}_0 s = \epsilon$. Since $\epsilon \in L_1$ and $\epsilon \in L_2$, it follows that $\epsilon \in L_1||L_2 \subseteq$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

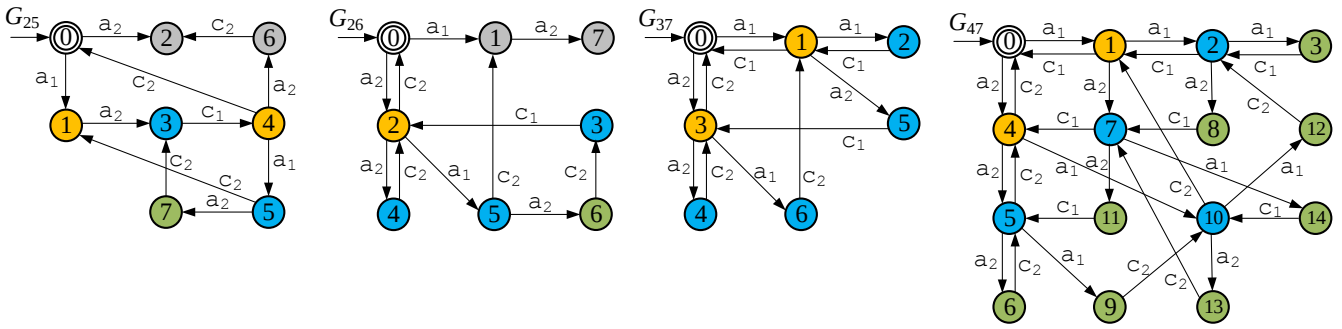

FIGURE 6: Composition of OP languages: Conflict in G_{25} and G_{26}; Supremal OP languages in G_{37} and G_{47}.

$\overline{L_1}||\overline{L_2}$. Hence, we have that $t_0 = \epsilon \in \overline{L_1}||\overline{L_2}$ and $s_0 = p(t_0) \in p(\overline{L_1}||\overline{L_2})$. For the induction step, we assume that $s_{k-1} = \mathrm{pre}_{k-1}s = \sigma_1 \cdots \sigma_{k-1}$ for some $1 \leq k \leq |s|$ and there is a $t_{k-1} \in \overline{L_1}||\overline{L_2}$ such that $p(t_{k-1}) = s_{k-1}$. Now consider $\sigma_k \in \Sigma$. Then, it holds that $t_{k-1} \in \overline{L_1}||\overline{L_2}$ and $s_{k-1} \in p(\overline{L_1}||\overline{L_2})$ and $\sigma_k \in \Sigma$ and $p(t_{k-1})\sigma_k = s_{k-1}\sigma_k = \mathrm{pre}_k s \in p(\overline{L_1}||\overline{L_2})$. According to Lemma 4, this implies that there is a $v_k \in (\Sigma_1 \cup \Sigma_2)^\star$ such that $p(v_k) = \sigma_k$ and $t_{k-1}v_k \in \overline{L_1}||\overline{L_2}$. In particular, for $k = |s|$, it follows that $t = t_0 v_1 \cdots v_{|s|} \in \overline{L_1}||\overline{L_2}$ and $s = \mathrm{pre}_{|s|}s = p(t) \in p(\overline{L_1}||\overline{L_2})$.

This concludes the induction step and we found that $s \in p(\overline{L_1}||\overline{L_2})$. It remains to show that $s \in p(L_1||L_2)$. To this end, we first conclude that $f(s) = \epsilon$ since $s \in L_C^{\mathrm{sup}}$ and $t \in s_1||s_2$ for some $s_1 \in \overline{L_1}$ and $s_2 \in \overline{L_2}$ since $t \in \overline{L_1}||\overline{L_2}$. Considering that $f(s) = m_1^{-1}(f_2(s_2))f_1(s_1)$ according to Lemma 3, it must be the case that $|f_2(s_2))| = |f_1(s_1)| = |f(s)| = 0$. Hence, $s_1 \in L_1$ and $s_2 \in L_2$, which implies that $t \in L_1||L_2$. Therefore, $s = p(t) \in p(L_1||L_2)$. Since $s \in L_C^{\mathrm{sup}}$ was arbitrary, it follows that $L_C^{\mathrm{sup}} \subseteq p(L_1||L_2)$, which concludes the proof of Theorem 4. □

The next theorem confirms the observation from the previous example. It holds that the composition of two supremal OP languages is non-conflicting.

Theorem 5. *Consider the setting in Theorem 4. Then, L_1 and L_2 are non-conflicting.*

Proof. Consider an arbitrary string $t \in \overline{L_1}||\overline{L_2}$. Then, it holds that $t \in s_1||s_2$ for some $s_1 \in \overline{L_1}$, $s_2 \in \overline{L_2}$ and $f(s) = \alpha_1 \cdots \alpha_j = m_1^{-1}(f_2(s_2))f_1(s_1)$ for $s = p(t)$ and some $0 \leq j \leq C = C_1 + C_2$. We further write $m_1^{-1}(f_2(s_2)) = \alpha_1 \cdots \alpha_l$ and $f_1(s_1) = \alpha_{l+1} \cdots \alpha_j$. We consider $\omega_i = m(\alpha_i) \in \Sigma^{\mathrm{out}}$ for $1 \leq i \leq j$ and define the string $v = \omega_1 \cdots \omega_l \gamma_{l+1} \omega_{l+1} \cdots \gamma_j \omega_j \in (\Sigma_1 \cup \Sigma_2)^\star$ for $\gamma_i = m_1(\alpha_i) = m_2^{-1}(\omega_i)$, $i = l+1, \ldots, j$. In addition, we write $v_1 = p_1(v) = \gamma_{l+1} \cdots \gamma_j$ and $v_2 = p_2(v) = v$. Since $f_1(s_1) = \alpha_{l+1} \cdots \alpha_j = m_1^{-1}(v_1)$ and $L_1 = L_{C_1}^{\mathrm{sup}}$, it holds that $s_1 v_1 \in L_1$. Furthermore, we first conclude that $m_1^{-1}(f_2(s_2 \omega_1 \cdots \omega_l)) = \epsilon$ since $m(m_1^{-1}(f_2(s_2))) = m_2(f_2(s_2)) = \omega_1 \cdots \omega_l$. That is, $f_2(s_2 \omega_1 \cdots \omega_l) = \epsilon$. Then, it directly follows that $f_2(s_2 \omega_1 \cdots \omega_l \gamma_{l+1} \omega_{l+1} \cdots \gamma_i \omega_i) = \epsilon$ for all $i = l+1, \ldots, j$. Hence, also $s_2 v_2 \in L_2$. Together, we have

$$tv \in (s_1||s_2)v \subseteq s_1 v_1||s_2 v_2 \subseteq L_1||L_2,$$

which implies that $t \in \overline{L_1||L_2}$. Since t was arbitrary, it follows that L_1 and L_2 are nonconflicting. □

The implications of Theorem 4 and 5 can be observed from G_{37} and G_{47} in Fig. 6 and the related automata in Fig. 3 and 5. According to the previous discussion, it is the case that $L_{\mathrm{m}}(G_{37}) = L_C^{\mathrm{sup}} = p(L_{\mathrm{m}}(G_3)||L_{\mathrm{m}}(G_7)) = p(L_{C_1}^{\mathrm{sup}}||L_{C_2}^{\mathrm{sup}})$ with $C_1 = 1$, $C_2 = 1$ and $C = C_1 + C_2 = 2$. Furthermore, G_{37} is nonblocking, which indicates that $L_{\mathrm{m}}(G_3)$ and $L_{\mathrm{m}}(G_7)$ are nonconflicting. Similarly, $L_{\mathrm{m}}(G_{47}) = L_C^{\mathrm{sup}} = p(L_{\mathrm{m}}(G_4)||L_{\mathrm{m}}(G_7)) = p(L_{C_1}^{\mathrm{sup}}||L_{C_2}^{\mathrm{sup}})$ with $C_1 = 2, C_2 = 1$ and $C = C_1 + C_2 = 3$ and G_{47} is nonblocking, Differently, $L_{\mathrm{m}}(G_{26}) \neq L_3^{\mathrm{sup}}$ since $L_{\mathrm{m}}(G_2) \neq L_2^{\mathrm{sup}}$ and $L_{\mathrm{m}}(G_6) \neq L_2^{\mathrm{sup}}$. In addition, G_{26} is blocking, that is, $L_{\mathrm{m}}(G_2)$ and $L_{\mathrm{m}}(G_6)$ are conflicting. Here, we further note that the conditions in Theorem 4 and 5 are sufficient but not necessary. Consider for example the OP languages $L_{\mathrm{m}}(G_8)$ in Fig. 7 and $L_{\mathrm{m}}(G_7)$ in Fig. 5. Then, it holds that $L_{\mathrm{m}}(G_8) \neq L_1^{\mathrm{sup}}$ but $L_{\mathrm{m}}(G_{87}) = L_{\mathrm{m}}(G_8)||L_{\mathrm{m}}(G_7) = L_2^{\mathrm{sup}}$ and $L_{\mathrm{m}}(G_8)$ and $L_{\mathrm{m}}(G_7)$ are nonconflicting.

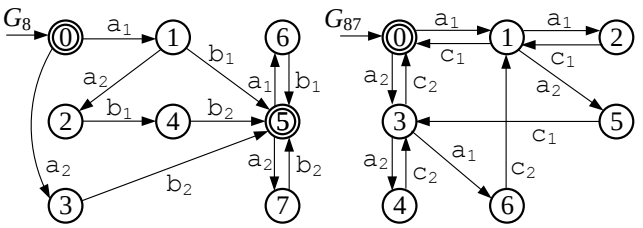

FIGURE 7: Composition of OP languages.

An important outcome of Theorem 4 is that the composition of two supremal OP languages is again a supremal OP language. That is, the result of the composition operation has the same properties as its arguments. Using this fact, we next state the following corollary, which shows that the composition of an arbitrary number of supremal OP languages

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

is again a supremal OP language and these languages are further nonconflicting.

Corollary 1. *Consider alphabets* Σ_i^{in}, Σ_i^{out}, Σ_i *such that* $\Sigma_i = \Sigma_i^{\text{in}} \cup \Sigma_i^{\text{out}}$ *and* $\Sigma_i^{\text{in}} \cap \Sigma_i^{\text{out}} = \emptyset$ *for* $i = 1, \ldots, n$ *and assume that* $\Sigma_i^{\text{out}} = \Sigma_{i+1}^{\text{in}}$ *for* $i = 1, \ldots, n-1$. *Further, let* $m_i : \Sigma_i^{\text{in}} \to \Sigma_i^{\text{out}}$ *be defined as in Section III-A. Assume that* $L_i = L_{C_i}^{\sup}(\Sigma_i^{\text{in}}, \Sigma_i^{\text{out}}, m_i)$ *with* $C_i > 0$ *for* $i = 1, \ldots, n$. *Write* $\Sigma^{\text{in}} = \Sigma_1^{\text{in}}$, $\Sigma^{\text{out}} = \Sigma_n^{\text{out}}$, $\Sigma = \Sigma^{\text{in}} \cup \Sigma^{\text{out}}$ *and let* $p : (\bigcup_{i=1}^n \Sigma_i)^\star \to \Sigma^\star$ *as well as* $m : \Sigma^{\text{in}} \to \Sigma^{\text{out}}$ *such that* $m(s) = m_n(\cdots m_1(s))$ *for any* $s \in (\Sigma^{\text{in}})^\star$. *Then,* $L = p(||_{i=1}^n L_i) = L_{C_1+\cdots+C_n}^{\sup}(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m)$ *and* $L_1, \ldots, L_n$ *are nonconflicting.*

Proof. We prove the corollary by induction. To this end, we introduce the alphabets $\Gamma_k = \Sigma_1^{\text{in}} \cup \Sigma_k^{\text{out}}$, $\Lambda_k = \bigcup_{i=1}^k \Sigma_i$ and $\Pi_k = \Sigma_1^{\text{in}} \cup \Sigma_k$ for $k = 2, \ldots, n$. We further use the natural projections $p_{\Lambda_k,\Gamma_k} : \Lambda_k^\star \to \Gamma_k^\star$, $p_{\Pi_k,\Gamma_k} : \Pi_k^\star \to \Gamma_k^\star$ and the languages $L_{1,k} = p_{\Lambda_k,\Gamma_k}(L_1||\cdots||L_k)$ for $k = 2, \ldots, n$.

For the initialization, we know that $L_{1,2} = p_{\Lambda_2,\Gamma_2}(L_1||L_2) = L_{C_1+C_2}^{\sup}$ from Theorem 4 and L_1, L_2 are nonconflicting from Theorem 5.

For the induction step, we assume that $L_{1,k} = L_{C_1+\cdots+C_k}^{\sup}$ and $L_1, \ldots, L_k$ are nonconflicting. Since $L_{1,k} = L_{C_1+\cdots+C_k}^{\sup}$ and $L_{k+1} = L_{C_{k+1}}^{\sup}$, it directly follows from Theorem 4 that

$$\begin{aligned}
&p_{\Lambda_{k+1},\Gamma_{k+1}}(L_1||\cdots||L_k||L_{k+1}) \\
=&p_{\Pi_{k+1},\Gamma_{k+1}}(p_{\Lambda_k,\Gamma_k}(L_1||\cdots||L_k)||L_{k+1}) \\
=&p_{\Pi_{k+1},\Gamma_{k+1}}(L_{1,k}||L_{k+1}) \\
=&p_{\Pi_{k+1},\Gamma_{k+1}}(L_{C_1+\cdots+C_k}^{\sup}||L_{C_{k+1}}^{\sup}) = L_{C_1+\cdots+C_{k+1}}^{\sup}.
\end{aligned}$$

With the same argument, Theorem 5 implies that $L_1||\cdots||L_k$ and L_{k+1} are non-conflicting. Hence, $L_1, \cdots, L_{k+1}$ are non-conflicting. Since k was arbitrary, it follows for $k = n-1$ that $L = p(||_{i=1}^n L_i) = L_{C_1+\cdots+C_n}^{\sup}$ and $L_1, \ldots, L_n$ are nonconflicting. □

In order to discuss the practical implication of Corollary 1, we consider an AMS with an arbitrary number of MCs that exchange products and that are modeled by supremal OP languages. Then, it holds that the composition of the MCs is again represented by the supremal OP language and the joint operation of the MCs is nonblocking.

D. USAGE OF COMPOSED ORDER-PRESERVING MODELS IN SUPERVISORY CONTROL

The previous section indicates that supremal OP languages are nonconflicting and their composition again leads to a supremal OP language. In this section, we determine a further important property of composed supremal OP languages in order to clarify their usability for the nonblocking supervisory control of large-scale DES. We first consider the case where a subsystem model of a larger DES is OP and is composed of two supremal OP languages $L_1 = L_{C_1}^{\sup}$ and $L_2 = L_{C_2}^{\sup}$ with the alphabets $\Sigma_1 = \Sigma_1^{\text{in}} \cup \Sigma_1^{\text{out}}$, $\Sigma_2 = \Sigma_2^{\text{in}} \cup \Sigma_2^{\text{out}}$ and $\Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$ as in Theorem 4. Then, the composed supremal OP language $L = p(L_1||L_2)$ is computed using $p : \hat{\Sigma}^\star \to \Sigma^\star$ with $\hat{\Sigma} = \Sigma_1 \cup \Sigma_2$ and $\Sigma = \Sigma_1^{\text{in}} \cup \Sigma_2^{\text{out}}$. This computation is illustrated in Fig. 8 using automata models G_1, G_2 and G such that $L_{\text{m}}(G_1) = L_1$, $L_{\text{m}}(G_2) = L_2$ and $L_{\text{m}}(G) = L$.

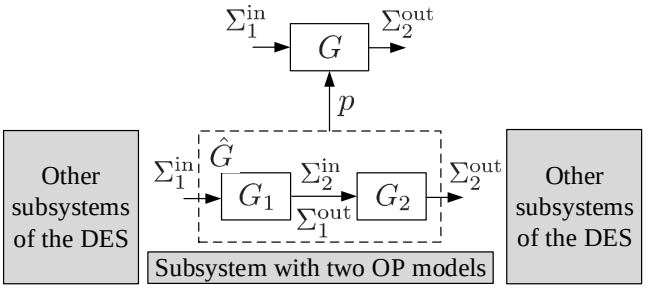

FIGURE 8: Subsystem with composition of two OP models.

Since the composed supremal OP language L is different from the composition of the original languages $L_1||L_2$, it needs to be verified if L can/should be used instead of $L_1||L_2$ when designing supervisors for the overall DES. In this context, it is important to note that the setting in Fig. 8 is directly related to the setting of abstraction-based supervisory control of large-scale modular DES [14], [15], [34]. Here, $\hat{G} = G_1||G_2$ with the alphabet $\hat{\Sigma}$ represents the original model of a subsystem of a larger DES and G with the alphabet Σ is the abstracted subsystem model. The abstraction alphabet $\Sigma = \Sigma_1^{\text{in}} \cup \Sigma_2^{\text{out}}$ is chosen such that it contains all the (external) events that are shared with other subsystems, whereas the internal events in $\Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$ are projected away. In this setting, the existing literature [15], [34] shows that a nonblocking supervisor computed for the system with the abstracted model G is as well nonblocking for the system with the original model $\hat{G}$ if the projection p fulfills certain sufficient conditions such as the natural observer condition in Definition 3 [51].

Definition 3. *Let* $L \subseteq \hat{\Sigma}^\star$ *be a language, and let* $p : \hat{\Sigma}^* \to \Sigma$ *be the natural projection for* $\Sigma \subseteq \hat{\Sigma}$. p *is a natural observer for* L *if it holds for all* $t \in \overline{L}$ *and* $u \in \Sigma^\star$ *that*

$$p(t)u \in p(L) \Rightarrow \exists v \in \hat{\Sigma}^\star \text{ s.t. } tv \in L \wedge p(v) = u.$$

In words, p is a natural observer for L if any string t that belongs to the prefix-closure $\overline{L}$ of L can be extended to a string in L whenever its projection $p(t)$ can be extended to a string in $p(L)$. We next show that the projection $p : (\Sigma_1 \cup \Sigma_2)^\star \to \Sigma^\star$, which is used for computing the composed supremal OP language $p(L_1||L_2)$ in the previous section, is a natural observer for $L_1||L_2$.

Theorem 6. *Assume that* Σ_1^{in}, Σ_1^{out}, Σ_2^{in}, Σ_2^{out}, m_1, m_2, Σ, m *and* p *are as introduced above with* $\Sigma^{\text{in}} = \Sigma_1^{\text{in}}$, $\Sigma^{\text{out}} = \Sigma_2^{\text{out}}$ *and* $\Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$. *Further, let* $L_1 = L_{C_1}^{\sup}$ *and* $L_2 = L_{C_2}^{\sup}$ *for* $C_1, C_2 > 0$. *Then,* $p : (\Sigma_1 \cup \Sigma_2)^\star \to \Sigma^\star$ *is a natural observer for* $L_1||L_2$.

Proof. We first recall that $\overline{L_1||L_2} = \overline{L}_1||\overline{L}_2$ from Theorem 5. In order to show that p is a natural observer for $L_1||L_2$, we

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

take an arbitrary string $t \in \overline{L_1||L_2} = \overline{L}_1||\overline{L}_2$ and $u \in \Sigma^\star$ such that $su = p(t)u \in p(L_1||L_2)$. We next show by induction that there is a $v \in (\Sigma_1 \cup \Sigma_2)^\star$ such that $p(v) = u$ and $tv \in L_1||L_2$. To this end, we write $u = \sigma_1 \cdots \sigma_{|u|}$ with $\sigma_k \in \Sigma$ for $k = 1, \ldots, |u|$. For the initialization, we note that there is a $v_1 \in (\Sigma_1 \cup \Sigma_2)^\star$ such that $tv_1 \in \overline{L}_1||\overline{L}_2$ and $p(v_1) = \sigma_1$ because of Lemma 4. For the induction step, we consider that $t_{k-1} = tv_1 \cdots v_{k-1} \in \overline{L}_1||\overline{L}_2$ such that $p(v_1 \cdots v_{k-1}) = \sigma_1 \cdots \sigma_{k-1}$. Applying Lemma 4, it directly follows that there exists a $v_k \in (\Sigma_1 \cup \Sigma_2)^\star$ with $p(v_k) = \sigma_k$ and such that $tv_1 \cdots v_k \in \overline{L}_1||\overline{L}_2$ and $p(v_1 \cdots v_k) = \sigma_1 \cdots \sigma_k$. Specifically, for $k = |u|$, it holds with $v = v_1 \cdots v_{|u|}$ that $tv \in \overline{L}_1||\overline{L}_2$ and $p(v) = \sigma_1 \cdots \sigma_{|u|}$. It remains to show that $tv \in L_1||L_2$. Since $su \in p(L_1||L_2)$, it must be the case that $|f(su)| = 0$. In addition, $tv \in \overline{L}_1||\overline{L}_2$ implies that $tv \in s_1||s_2$ for some $s_1 \in \overline{L}_1$, $s_2 \in \overline{L}_2$. Then, according to Lemma 3, we have that $f(s) = m_1^{-1}(f_2(s_2))f_1(s_1)$ and hence, $|f_2(s_2)| + |f_1(s_1)| = 0$. Accordingly, $f_2(s_2) = f_1(s_1) = \epsilon$, which shows that $s_1 \in L_1$ and $s_2 \in L_2$. Therefore, $tv \in s_1||s_2 \subseteq L_1||L_2$, which concludes the proof. □

According to the previous discussion, Theorem 6 implies that the abstracted model G with the marked language $L_\mathrm{m}(G) = L$ can be used for the nonblocking supervisor synthesis instead of the original model $\hat{G} = G_1||G_2$. Hereby, the main advantage is that G cannot have more states (and generally has fewer states) than $\hat{G}$ since p is a natural observer for $L_1||L_2$ [51]. In summary, when computing nonblocking supervisors for modular DES with OP subsystems, it is possible to use the composed OP model with a smaller state count than the composition of the original OP models in order to reduce the computational effort.

We illustrate the discussed features by the example automaton G_{47} in Fig. 6 that is computed such that $L_\mathrm{m}(G_{47}) = p(L_\mathrm{m}(G_4||G_7))$ with G_4 in Fig. 3 and G_7 in Fig. 5. Assuming that G_4 and G_7 are components of a modular system, it is not required to use the automaton $G_4||G_7$ with 21 states for the nonblocking supervisor synthesis. Instead, it is possible to use G_{47} in Fig. 6 with only 15 states.

We note that the result in Theorem 6 was stated for the case of two OP languages. We next extend this result to the case of an arbitrary number of supremal OP languages similar to Corollary 1. The extended setting is illustrated in Fig. 9 with the automata G_i such that $L_\mathrm{m}(G_i) = L_i$ for $i = 1, \ldots, n$.

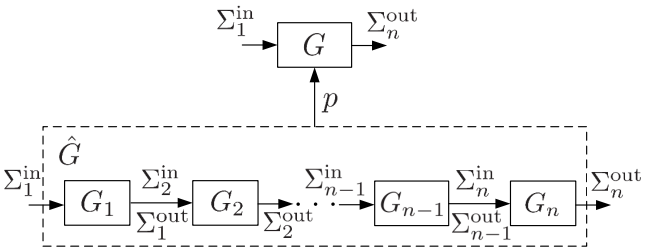

FIGURE 9: Subsystem with n OP models.

Corollary 2. *Assume that* Σ_i^{in}, Σ_i^{out}, m_i, $i = 1, \ldots, n$ *are introduced as above with* $\Sigma_i^{\mathrm{out}} = \Sigma_{i+1}^{\mathrm{in}}$ *for* $i = 1, \ldots, n-1$. *Further, define* $m = m_n(\cdots m_2(m_1))$ *and* $\Sigma = \Sigma_1^{\mathrm{in}} \cup \Sigma_n^{\mathrm{out}}$. *Then,* $p : (\Sigma_1 \cup \cdots \cup \Sigma_n)^\star \to \Sigma^\star$ *is a natural observer for* $L_1|| \cdots ||L_n$.

We prove Corollary 2 in Appendix E. As the main implication of Corollary 2, it is possible to use the automaton $\hat{G}$ with $L_\mathrm{m}(\hat{G}) = p(L_1|| \cdots ||L_n)$ with a smaller state count than the automaton G with $L_\mathrm{m}(G) = L_1|| \cdots ||L_2$ for the nonblocking supervisor computation.

IV. FLEXIBLE MANUFACTURING SYSTEM APPLICATION

In this section, we apply the proposed modeling framework to the FMS example in [10], [23]. This example is chosen since it offers different product types that share robots and machines with a capacity greater than one. We describe the FMS and provide OP models of the relevant MCs and a supervisor design in Section IV-A. Section IV-B and IV-C perform modifications of the FMS that illustrate the advantage of composing OP models and that demonstrate the supervisor design for AMSs with sequential operations and assembly operations, respectively. Finally, Section IV-D elaborates on several practical considerations when using OP models.

A. FMS DESCRIPTION

The outline of the FMS in [10], [23] is shown in Fig. 10. It consists of 3 robots R1, R2, R3 and 4 machines M1, M2, M3, M4. The robots are able to transport products from/to the machines. In addition, the robots can take products from the input buffers I1, I2, I3 and deliver them to the output buffers O1, O2, O3.

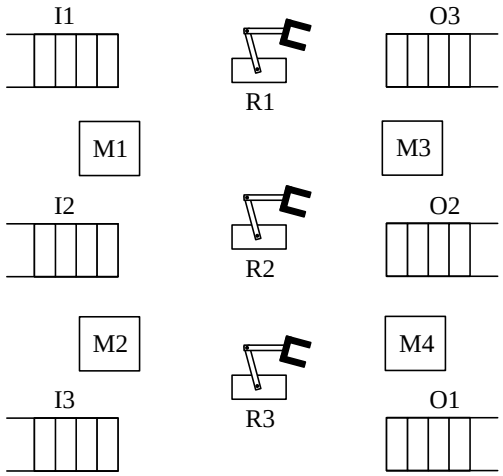

FIGURE 10: FMS overview.

In principle, it is desired to process 3 different product types using the FMS in Fig. 10. For the first product type, R2 takes a product from I1 and moves it to M2. Then, R2 delivers the product from M2 to O1. The product path is shown in Fig. 11 (a). The second product type is taken from I2 by R3 and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

transported to M4. Then, R2 moves the product to M3 and R1 delivers the product to O2 as can be seen in Fig. 11 (b). The third product type has two alternative paths that start from I3. On the first path, R1 moves the product to M3, R2 moves the product to M4 and R3 delivers the product to O3. On the second path, R1 moves the product to M1, R2 moves the product to M2 and R3 delivers the product to O3 as indicated in Fig. 11 (c).

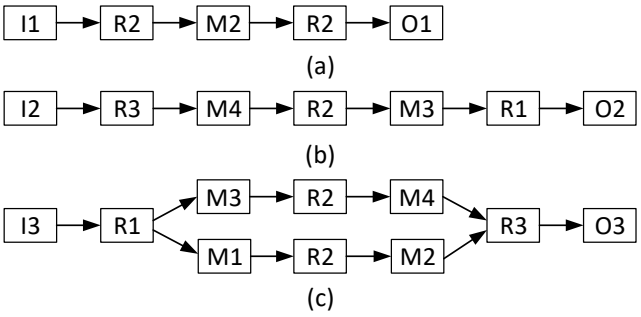

FIGURE 11: Product paths through the FMS.

According to the FMS setup, it is readily observed that several MCs share different product types. For example, R1 moves P2 and P3, R2 transports P1, P2, P3 and R3 delivers P2, P3. In addition, M2 processes P1, P3, M3 processes P2, P3 and M4 processes P2, P3. Hence, it is clear that OP models are needed for R1, R2, R3, M2, M3 and M4. Following the description in [10], the robots can hold a single product, whereas the machines are able to hold up to two products simultaneously. That is, a simple OP model with capacity 1 as in Fig. 12 is suitable to represent the robots. Hereby, we use the following convention for the event names: each event name captures the name of the MC, where the product comes from, the name of the MC that holds the product after delivery and the product type. For example, the name for the event that characterizes moving a product of type P3 from I3 to R1 is written as $\texttt{I3-R1}_{\texttt{P3}}$.

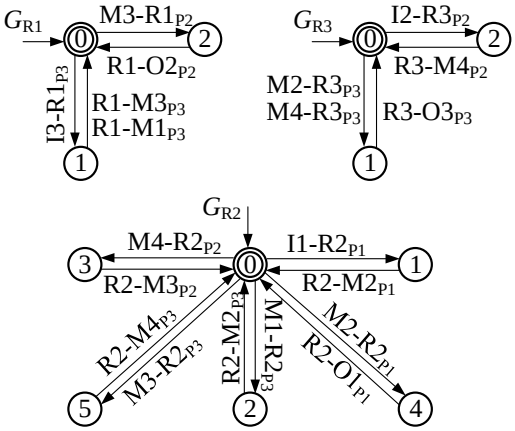

FIGURE 12: Robot models.

Different from the robots, an OP model with capacity 2 is needed to capture the behavior of M2, M3 and M4 as is shown in Fig. 13. Since there are two different product types on each of these machines, the resulting model has 7 states. Note that no OP model is needed for M1 since this machine only processes a single product type.

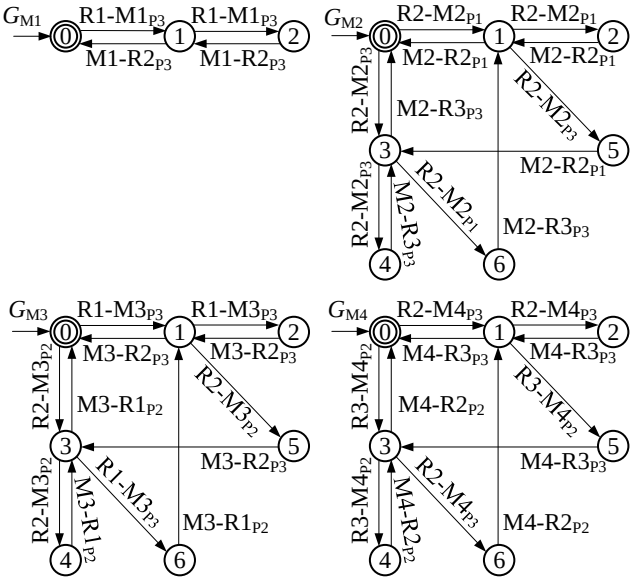

FIGURE 13: OP machine models.

Finally, [10] provides models for the input and output buffers that limit the numbers of products of a certain type that can enter the FMS to 3 for P1, 7 for P2 and 11 for P3. These buffer models are shown in Fig. 14.

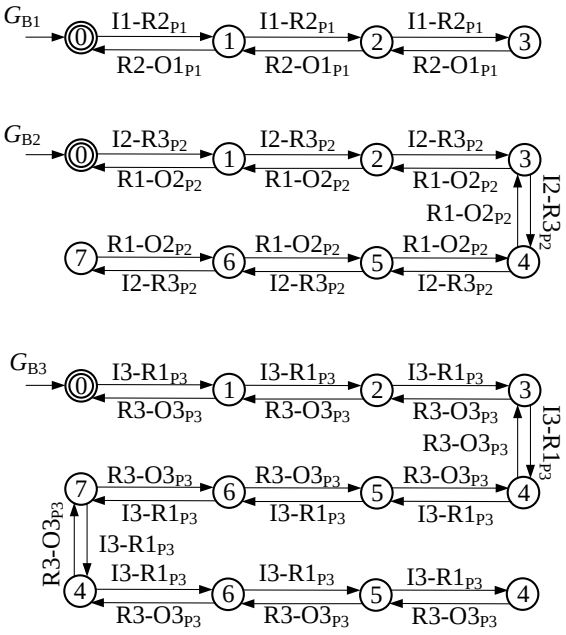

FIGURE 14: Buffer models.

In this context, we note that [10] does not employ OP

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

models for M2, M3, M4. The respective models in [10] are shown in Fig. 15. That is, these models assume that, even a certain product type (such as P1 on M2) enters M2 before P3, it is possible that P3 leaves M2 first. In practice this means that a product can overtake another product when moving through the FMS. Although this might be possible, it is then necessary to determine the type of each product leaving such machine (since the event indicating the departure of a product is directly related to the product type). This requires additional sensor information and implementation effort. Differently, the OP models in Fig. 13 naturally keep track of the products entering and leaving each machine.

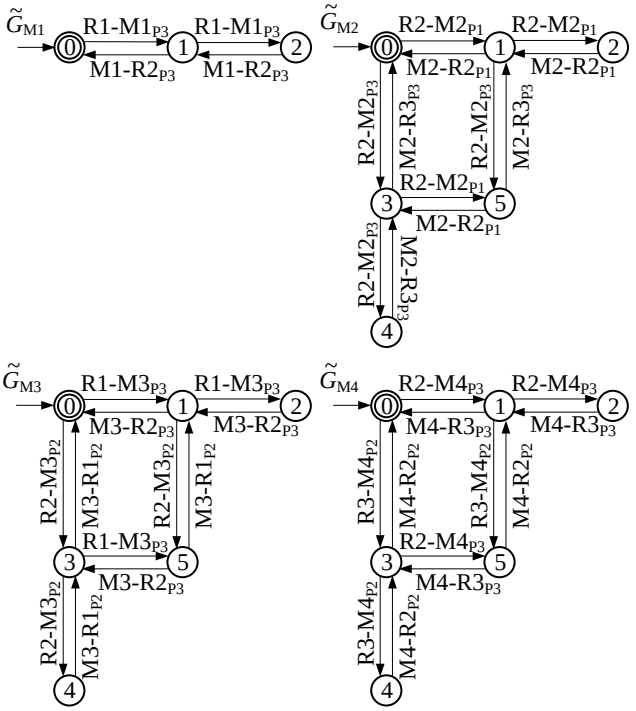

FIGURE 15: Machine models.

We next discuss the supervisor computation for the FMS according to Section II-B. To this end, we first note that the aim of this paper is not an efficient supervisor design method but the usage of OP models as introduced in Section III. In addition, it holds that the product paths are already encoded in the MC models such that no additional specification is required. That is, we compute the overall plant as

$$G = G_{\mathrm{R1}}||G_{\mathrm{R2}}||G_{\mathrm{R3}}||G_{\mathrm{M1}}||G_{\mathrm{M2}}||G_{\mathrm{M3}}||G_{\mathrm{M4}}||G_{\mathrm{B1}}||G_{\mathrm{B2}}||G_{\mathrm{B3}}.$$

Then, we compute a nonblocking supervisor S such that $L_{\mathrm{m}}(S||G) = \mathrm{SupC}(L_{\mathrm{m}}(G), G, \Sigma_{\mathrm{u}})$ with $\Sigma_{\mathrm{u}} = \emptyset$ with 30 692 states. In comparison, the nonblocking supervisor $\tilde{S}$ for the original model

$$\tilde{G} = G_{\mathrm{R1}}||G_{\mathrm{R2}}||G_{\mathrm{R3}}||\tilde{G}_{\mathrm{M1}}||\tilde{G}_{\mathrm{M2}}||\tilde{G}_{\mathrm{M3}}||\tilde{G}_{\mathrm{M4}}||G_{\mathrm{B1}}||G_{\mathrm{B2}}||G_{\mathrm{B3}}.$$

in [10] (that is not OP) has 27 770 states. In addition, it can be verified that $L_{\mathrm{m}}(S||G) \subseteq L_{\mathrm{m}}(\tilde{S}||\tilde{G})$. This is due to the fact that the machine models in [10] do not preserve the order of products and hence model unnecessary behavior.

B. COMPOSED ORDER-PRESERVING MODELS FOR THE FMS

In order to illustrate the benefits of composing OP models as described in Section III-D, we consider a slightly modified version of the FMS in Section IV-A. To this end, we replace the machines M2, M3, M4 by corresponding workcells W2, W3, W4 that consist of two CBs and one machine. The outline of these workcells is shown in Fig. 16.

FIGURE 16: Workcell outline.

That is, products enter the workcell from the input CB (CB2i, CB3i, CB4i), are then processed by the respective machine (M2, M3, M4) and leave the workcell from the output CB (CB2o, CB3o, CB4o). Assuming a capacity of 1 for each of the MCs in the workcells, we use the following OP models in Fig. 17.

Then, we first determine the state sizes of the original and reduced workcell models. We write

$$\hat{G}_{\mathrm{W2}} = G_{\mathrm{CB2i}}||G_{\mathrm{M2}}||G_{\mathrm{CB2o}},$$
$$\hat{G}_{\mathrm{W3}} = G_{\mathrm{CB3i}}||G_{\mathrm{M3}}||G_{\mathrm{CB3o}},$$
$$\hat{G}_{\mathrm{W4}} = G_{\mathrm{CB4i}}||G_{\mathrm{M4}}||G_{\mathrm{CB4o}}$$

for the original workcell models with the alphabets $\hat{\Sigma}_{\mathrm{W2}} = \Sigma_{\mathrm{CB2i}} \cup \Sigma_{\mathrm{M2}} \cup \Sigma_{\mathrm{CB2o}}$, $\hat{\Sigma}_{\mathrm{W3}} = \Sigma_{\mathrm{CB3i}} \cup \Sigma_{\mathrm{M3}} \cup \Sigma_{\mathrm{CB3o}}$ and $\hat{\Sigma}_{\mathrm{W4}} = \Sigma_{\mathrm{CB4i}} \cup \Sigma_{\mathrm{M4}} \cup \Sigma_{\mathrm{CB4o}}$. According to Section III-D, we also introduce the alphabets of the reduced OP models for the workcells as $\Sigma_{\mathrm{W2}} = \Sigma_{\mathrm{W2}} \setminus \Sigma_{\mathrm{M2}}$, $\Sigma_{\mathrm{W3}} = \Sigma_{\mathrm{W3}} \setminus \Sigma_{\mathrm{M3}}$ and $\Sigma_{\mathrm{W4}} = \Sigma_{\mathrm{W4}} \setminus \Sigma_{\mathrm{M4}}$ with the related natural projections $p_{\mathrm{W2}} : \hat{\Sigma}^{\star}_{\mathrm{W2}} \to \Sigma^{\star}_{\mathrm{W2}}$, $p_{\mathrm{W3}} : \hat{\Sigma}^{\star}_{\mathrm{W3}} \to \Sigma^{\star}_{\mathrm{W3}}$ and $p_{\mathrm{W4}} : \hat{\Sigma}^{\star}_{\mathrm{W4}} \to \Sigma^{\star}_{\mathrm{W4}}$. That is, the reduced models only contain the external events of the workcells that are shared with the robots. Then, we write G_{W2}, G_{W3} and G_{W4} for the reduced OP models such that $L_{\mathrm{m}}(G_{\mathrm{W2}}) = p_{\mathrm{W2}}(L_{\mathrm{m}}(\hat{G}_{\mathrm{W2}}))$, $L_{\mathrm{m}}(G_{\mathrm{W3}}) = p_{\mathrm{W3}}(L_{\mathrm{m}}(\hat{G}_{\mathrm{W3}}))$ and $L_{\mathrm{m}}(G_{\mathrm{W4}}) = p_{\mathrm{W4}}(L_{\mathrm{m}}(\hat{G}_{\mathrm{W4}}))$.

It holds that $\hat{G}_{\mathrm{W2}}$, $\hat{G}_{\mathrm{W3}}$ and $\hat{G}_{\mathrm{W4}}$ have 27 states, whereas G_{W2}, G_{W3} and G_{W4} have only 15 states (which complies with (10)). Accordingly, a nonblocking supervisor for the overall system with the original workcell models $\hat{G}_{\mathrm{W2}}$, $\hat{G}_{\mathrm{W3}}$ and $\hat{G}_{\mathrm{W4}}$ has 2 444 288 states, whereas a nonblocking supervisor for the overall system with the reduced workcell models G_{W2}, G_{W3} and G_{W4} has 643 100 states. That is, in addition to correctly modeling the order of products traveling to the FMS, it is possible to reduce the size of the required models using the particular properties of composed OP languages.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

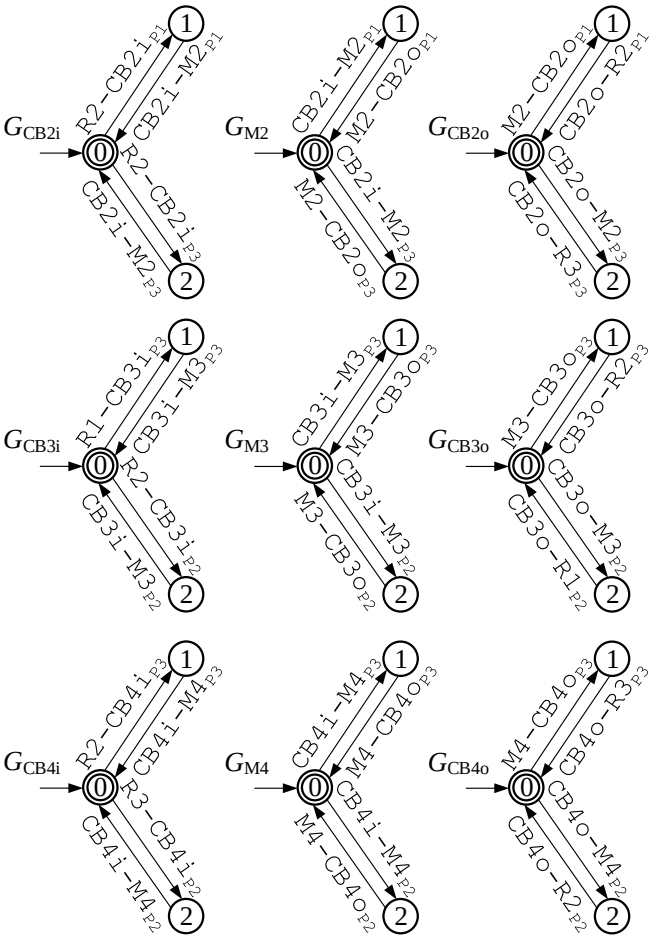

FIGURE 17: Workcell models.

C. COMBINATION WITH ASSEMBLY OPERATIONS

The previous discussion focuses on sequential operations in AMSs in order to illustrate the concept of OP models. In general, it is the case that AMSs comprise both sequential operations, which characterize the flow of products through the AMS, and assembly operations, where parts of a product are combined in order to obtain a final product. The existing literature considers the supervisory control for AMSs with the combination of sequential and assembly operations both using automata models [9], [52], [53] and Petri Net models [4], [54] but without including the order of products in the model. We next illustrate the usage of OP models as introduced in this paper for AMSs that both include assembly operations and sequential operations that preserve the product order. To this end, we employ a modification of the product paths in Fig. 11 as shown in Fig. 18. Here, P1 and P2 are considered as parts of a final product that enter from the input buffers I1 and I2, respectively. After following their respective production paths, these two product parts are assembled in the output assembly station OA. That is, the product order is preserved during the sequential operations along the product paths, whereas P1 and P2 are combined when arriving at OA.

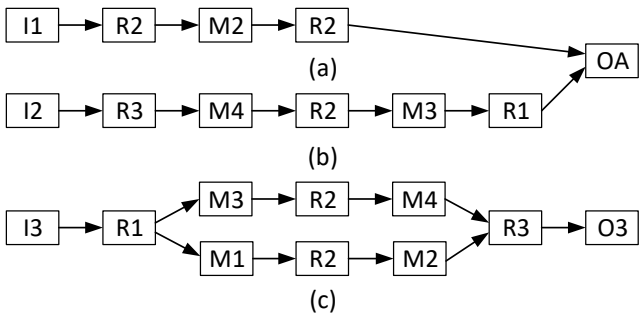

FIGURE 18: Modified FMS with assembly operation.

The same automata models as in Section IV-A are used for the MCs I3, O3, R3, M1, M2, M3 and M4. The robot models G_{R1}, G_{R2} and the buffer models G_{B1}, G_{B2} are modified by replacing the events `R1-O2_P2` and `R2-O1_P1` by `R1-OA_P2` and `R2-OA_P1`, respectively, to indicate that the product parts P1 and P2 are delivered to OA (see Fig. 19). In addition, the model G_{OA} of OA in Fig. 19 captures that the product parts P1 and P2 can enter OA in an arbitrary order (events `R1-OA_P2` and `R2-OA_P1`). When both product parts are present, their assembly and discharge from the AMS is modeled by the event `P1-P2`.

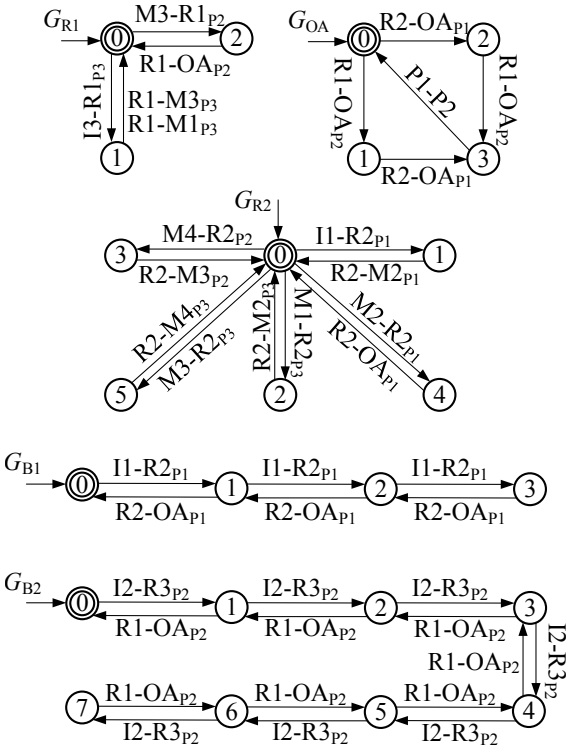

FIGURE 19: Modified MC models including an assembly operation.

In analogy to Section IV-A, we computed a nonblocking supervisor S for the modified FMS including the assembly operation in Fig. 18. The supervisor has 120 697 states addresses both the OP sequential operations and the assembly operation of the FMS.

D. PRACTICAL CONSIDERATIONS

In this section, we further clarify the main contribution of this paper. To this end, we identify several important design problems, where the proposed OP model can be used and we discuss possible directions for future research.

The main focus of this paper is the development of a general automaton-based OP model for sequential processes in AMSs. This model can be used for any MC of an AMS, where the order of products entering and leaving the MC has to be preserved. In this context, we point out that we do not require that all MCs of an AMS are OP. Since the proposed OP model is represented by a finite state automaton, it can be used together with automata-based models of MCs that are not OP. This is for example the case in Section IV-C, where the modified FMS comprises both sequential operations and assembly operations. That is, our OP model is complementary to existing models for MCs.

This paper is concerned with the development of OP models and the analysis of their properties. Due to this reason, the previous sections consider medium-scale AMSs and perform a monolithic supervisor design in order to demonstrate the usability of OP models. In practice, large-scale AMSs require the application of efficient supervisor design methods [14], [15], [24], [29]–[34], [55]. Since the proposed OP model is represented by a finite state automaton, any of the automata-based supervisor design methods for large-scale systems including for example the compositional synthesis in [33] and the abstraction-based design in [14], [15], [34], [55] can be applied to AMSs including OP models. In addition, an interesting direction for future research is the development of OP models for Petri Nets in order to benefit from recent efficient design methods and simplified supervisor representations as in [24], [29]–[32].

The OP models proposed in this paper are developed for the supervisory control of AMSs, which ensures the correct logical system behavior based on a given behavioral specification. In practice, the temporal system behavior also needs to be taken into account in order to achieve a satisfactory system performance such as small cycle time and high throughput [56]–[60]. Since the proposed OP model is represented by a finite state automaton, any of the automata-based performance-oriented design methods such as the directed control in [56], the time-optimal control based on sequential abstraction in [58] and the time-optimal control using compositional optimization in [59] can be applied to AMSs including OP models. In addition, developing OP models for Petri Nets will make it possible to apply Petri Net-based supervisor synthesis methods for performance improvement such as [57].

V. CONCLUSIONS

The subject of this paper is the development of discrete event system (DES) models for a particular property of automated manufacturing systems (AMSs). The model accounts for the fact that AMSs are able to manufacture different product types that potentially share various manufacturing components (MCs) such as machines and robots. Different from the existing literature, the proposed model captures the general case, where MCs can hold multiple products and then process these products sequentially. Specifically, if the capacity of a MC is greater than one, products will leave the MC in the order of entering the MC. That is, the MC needs to keep track of the product order if different products require different processing steps after leaving the MC.

As the main contribution, we formalize this scenario by introducing the new class of order-preserving (OP) languages. We show that a supremal OP language exists and can be recognized by a finite state automaton. In addition, we prove that it is possible to compose supremal OP languages to again obtain a supremal OP language. We further show that reduced models of composed OP languages that are suitable for nonblocking supervisory control can be obtained. We demonstrate the practicability of the proposed model by applying it to the supervisory control of a flexible manufacturing system. We further show that OP models can be used together with non-OP models for AMSs with both assembly operations and sequential operations that preserve the product order.

In future work we will study nonblocking modular system behavior of OP AMSs depending on the connectivity of MCs. In addition, we intend to investigate the development of OP models for Petri Nets in order to benefit from recent advances in the efficient supervisor synthesis.

.

APPENDIX A PROOF OF LEMMA 1

We provide the proof of Lemma 1.

Proof. We have to show that (i) $s \in L_{\mathrm{m}}(G_C^{\mathrm{sup}}) \Rightarrow s$ is OP and $c(s) \leq C$, (ii) $s \in L_C^{\mathrm{sup}} \Rightarrow s \in L_{\mathrm{m}}(G_C^{\mathrm{sup}})$.

For (i), we use induction on the string length to show that for each prefix $s_k := \mathrm{pre}_k(s) \leq s$, $p^{\mathrm{out}}(s_k) \leq m(p^{\mathrm{in}}(s_k))$, $c(s_k) \leq C$ and $\delta_C^{\mathrm{sup}}(x_{0,C}^{\mathrm{sup}}, s_k) = [f(s_k)]$.

Let $s \in L_{\mathrm{m}}(G_C^{\mathrm{sup}})$ be arbitrary. For the initialization, we consider $s_0 = \epsilon \leq s$. Then, it holds that $p^{\mathrm{out}}(s_0) = m(p^{\mathrm{in}}(s_0)) = \epsilon$ and $c(s_0) = 0 \leq C$. Furthermore, $\delta_C^{\mathrm{sup}}(x_0, s_0) = [f(s_0)] = [\epsilon]$.

Now assume that for some $k < |s|$, we have that $p^{\mathrm{out}}(s_k) \leq m(p^{\mathrm{in}}(s_k))$, $c(s_k) \leq C$ and $\delta_C^{\mathrm{sup}}(x_0, s_k) = [f(s_k)]$. We show that also $p^{\mathrm{out}}(s_{k+1}) \leq m(p^{\mathrm{in}}(s_{k+1}))$, $c(s_{k+1}) \leq C$ and $\delta_C^{\mathrm{sup}}(x_0, s_{k+1}) = [f(s_{k+1})]$.

There are three cases. In the first case, let $|f(s_k)| = 0$. That is, $[f(s_k)] = [\epsilon]$. Then, $s_{k+1} = s_k\,\alpha_1$ with $\alpha_1 \in \Sigma^{\mathrm{in}}$ according to (R1). Furthermore, it holds that $p^{\mathrm{out}}(s_{k+1}) = p^{\mathrm{out}}(s_k\alpha_1) = p^{\mathrm{out}}(s_k) \leq m(p^{\mathrm{in}}(s_{k+1})) = m(p^{\mathrm{in}}(s_k\alpha_1)) = m(p^{\mathrm{in}}(s_k))m(\alpha_1)$, $c(s_{k+1}) = c(s_k\alpha_1) = \max\{c(s_k), 1\} \leq C$ and $\delta_C^{\mathrm{sup}}(x_{0,C}^{\mathrm{sup}}, s_{k+1}) = \delta_C^{\mathrm{sup}}([\epsilon], \alpha_1) = [\alpha_1] = [f(s_{k+1})]$. In the second case, let $|f(s_k)| = C$. That is, $[f(s_k)] = [\alpha_1\alpha_2\cdots\alpha_C]$ for some $\alpha_1, \ldots, \alpha_C \in \Sigma^{\mathrm{in}}$. Then, $s_{k+1} = s_k\,\omega_1$ with $\omega_1 = m(\alpha_1) \in \Sigma^{\mathrm{out}}$ according to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

(R4). Noting that $f(s_k) = \alpha_1\alpha_2\cdots\alpha_C$, we further conclude that

$$\begin{aligned} p^{\text{out}}(s_{k+1}) &= p^{\text{out}}(s_k\omega_1) = p^{\text{out}}(s_k)\omega_1 \\ &= m(\text{pre}_{|p^{\text{out}}(s_k)|}(p^{\text{in}}(s_k)))\omega_1 \\ &= m(\text{pre}_{|p^{\text{out}}(s_k)|}(p^{\text{in}}(s_k)))m(\alpha_1) \\ &\leq m(\text{pre}_{|p^{\text{out}}(s_k)|}(p^{\text{in}}(s_k)))m(\alpha_1\alpha_2\cdots\alpha_C) \\ &= m\big(\text{pre}_{|p^{\text{out}}(s_k)|}(p^{\text{in}}(s_k))\text{suf}_{|p^{\text{out}}(s_k)|}(p^{\text{in}}(s_k))\big) \\ &= m(p^{\text{in}}(s_k)) = m(p^{\text{in}}(s_k\omega_1)) = m(p^{\text{in}}(s_{k+1})). \end{aligned}$$

Furthermore, $f(s_{k+1}) = f(s_k\omega_1) = \alpha_2\cdots\alpha_C$ and accordingly $\delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s_{k+1}) = \delta_C^{\text{sup}}([\alpha_1\alpha_2\cdots\alpha_C], \omega_1) = [\alpha_2\cdots\alpha_C] = [f(s_{k+1})]$ with (R4). Finally, since $c(s_k) \leq C$ and $c(s_{k+1}) \leq c(s_k)$, it follows that $c(s_{k+1}) \leq C$. In the third case, $0 < |f(s_k)| < C$. That is, $[f(s_k)] = [\alpha_1\cdots\alpha_j]$ with $j < C$. Then, there are two possible transitions. First, let $\sigma = \alpha_{j+1} \in \Sigma^{\text{in}}$ according to (R3). Then, the same argument as in the first case shows that $p^{\text{out}}(s_{k+1}) = p^{\text{out}}(s_k\alpha_{j+1}) = p^{\text{out}}(s_k) \leq m(p^{\text{in}}(s_{k+1})) = m(p^{\text{in}}(s_k\alpha_{j+1})) = m(p^{\text{in}}(s_k))m(\alpha_{j+1})$, $c(s_{k+1}) = c(s_k\alpha_{j+1}) = \max\{c(s_k), |f(s_k)| + 1\} \leq C$ and $\delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s_{k+1}) = \delta_C^{\text{sup}}([\alpha_1\cdots\alpha_j], \alpha_{j+1}) = [\alpha_1\cdots\alpha_j\alpha_{j+1}] = [f(s_{k+1})]$. Second, let $\sigma = \omega_1 = m(\alpha_1) \in \Sigma^{\text{out}}$ according to (R2) or (R4). Then, the same argument as in the second case shows that $p^{\text{out}}(s_{k+1}) \leq m(p^{\text{in}}(s_{k+1}))$, $f(s_{k+1}) = \alpha_2\cdots\alpha_j$ and accordingly $\delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s_{k+1}) = \delta_C^{\text{sup}}([\alpha_1\alpha_2\cdots\alpha_j], \omega_1) = [\alpha_2\cdots\alpha_j] = [f(s_{k+1})]$.

Since k was arbitrary, it follows for $k+1 = |s|$ that $p^{\text{out}}(s) \leq m(p^{\text{in}}(s))$, $c(s) \leq C$ and $\delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s) = [f(s)]$. Considering that $s \in L_{\text{m}}(G_C^{\text{sup}})$, it is further the case that $\delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s) = [\epsilon]$. Hence, it is the case that $m(p^{\text{in}}(s)) = p^{\text{out}}(s)$. That is, s is OP according to Definition 1.

For (ii), we consider an arbitrary string $s \in L_C^{\text{sup}}$ and we show by induction that it holds for each $k \leq |s|$ that $s_k \in L(G_C^{\text{sup}})$ and $\delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s_k) = [f(s_k)]$ in order to show that $s \in L_{\text{m}}(G_C^{\text{sup}})$. For initialization, we consider $s_0 = \epsilon$. Then, it holds that $\delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, \epsilon) = [\epsilon] = [f(s_0)]$. For the induction step, we let $k < |s|$ and assume that $\delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s_k) = [f(s_k)]$. Then, we show that $\delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s_{k+1}) = [f(s_{k+1})]$. In general, $s_{k+1} = s_k\sigma$ for $\sigma \in \Sigma$ and there are different cases for σ. First, consider that $f(s_k) = \epsilon$. Since s is OP, this implies that $p^{\text{out}}(s_k) = m(p^{\text{in}}(s_k))$. Hence, it must be the case that $\sigma = \alpha_1 \in \Sigma^{\text{in}}$. But then, (R1) ensures that $\delta_C^{\text{sup}}([\epsilon], \alpha_1) = \delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s_k\alpha_1) = \delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s_{k+1}) = [\alpha_1] = [f(s_{k+1})]$. Second, consider that $f(s_k) = \alpha_1\alpha_2\cdots\alpha_C$. Since the capacity of s is $c(s) = C$ and s is OP, it must be the case that $\sigma = \omega_1$ for $\omega_1 = m(\alpha_1)$. But then, (R4) ensures that $\delta_C^{\text{sup}}([\alpha_1\alpha_2\cdots\alpha_C], \omega_1) = \delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s_k\omega_1) = \delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s_{k+1}) = [\alpha_2\cdots\alpha_C] = [f(s_{k+1})]$. Third, consider that $f(s_k) = \alpha_1\alpha_2\cdots\alpha_j$ for $0 < j < C$. Since $0 < |f(s_k)| < C$ and s is OP, it is possible that $\sigma = \alpha_{j+1}$ for some $\alpha_{j+1} \in \Sigma^{\text{in}}$ or $\sigma = \omega_1 = m(\alpha_1) \in \Sigma^{\text{out}}$. If $\sigma = \alpha_{j+1}$, (R3) ensures that $\delta_C^{\text{sup}}([\alpha_1\cdots\alpha_j], \alpha_{j+1}) = \delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s_k\alpha_{j+1}) = \delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s_{k+1}) = [\alpha_1\cdots\alpha_j\alpha_{j+1}] = [f(s_{k+1})]$. If $\sigma = \omega_1$, (R2) or (R4) ensure that $\delta_C^{\text{sup}}([\alpha_1\alpha_2\cdots\alpha_j], \omega_1) = \delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s_k\omega_1) = \delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s_{k+1}) = [\alpha_2\cdots\alpha_j] = [f(s_{k+1})]$.

In particular, for $k+1 = |s|$, we have that $\delta_C^{\text{sup}}(x_{0,C}^{\text{sup}}, s) = [f(s)] = [\epsilon] = x_{0,C}^{\text{sup}}$ since s is OP and, hence, $|p^{\text{in}}(s)| = |p^{\text{out}}(s)|$. That is, $s \in L_{\text{m}}(G_C^{\text{sup}})$. □

APPENDIX B PROOF OF LEMMA 2

This section provides the proof of Lemma 2. For brevity, we write $L_{C_1}^{\text{sup}}$ and $L_{C_2}^{\text{sup}}$ instead of $L_{C_1}^{\text{sup}}(\Sigma_1^{\text{in}}, \Sigma_1^{\text{out}}, m_1)$ and $L_{C_2}^{\text{sup}}(\Sigma_2^{\text{in}}, \Sigma_2^{\text{out}}, m_2)$.

Proof. We have to show that $c(L) = C = C_1 + C_2$. Since $L_1 = L_{C_1}^{\text{sup}}$ and $L_2 = L_{C_2}^{\text{sup}}$ for $C_1, C_2 > 0$, it holds that $\epsilon \in L_1||L_2 \neq \emptyset$. Hence, we already know from Theorem 3 that $C = c(L) \leq C_1 + C_2$. In order to show that $C = C_1 + C_2$, it remains to show that $C \geq C_1 + C_2$.

We consider two cases. In the first case $C_2 \leq C_1$. In addition, for simplicity, we assume that $C_1 \leq 2\,C_2$. The extension for the case $C_1 > 2\,C_2$ is conceptually straightforward. Since the capacity of L_1 is $c(L_1) = C_1$ and $L_1 = L_{C_1}^{\text{sup}}$, it holds for an arbitrary $\alpha_1 \in \Sigma_1^{\text{in}}$ and $\omega_1 = m_1(\alpha_1) \in \Sigma_1^{\text{out}}$ that

$$s_1 = \underbrace{\alpha_1\cdots\alpha_1}_{C_2}\underbrace{\omega_1\cdots\omega_1}_{C_2}\underbrace{\alpha_1\cdots\alpha_1}_{C_1}\underbrace{\omega_1\cdots\omega_1}_{C_2}\underbrace{\omega_1\cdots\omega_1}_{C_1-C_2} \in L_1$$

and $c(s_1) = C_1$. Now consider that $\alpha_2 = \omega_1 \in \Sigma_2^{\text{in}} = \Sigma_1^{\text{out}}$ and $\omega_2 = m_2(\alpha_2) \in \Sigma_2^{\text{out}}$. Since the capacity of L_2 is $c(L_2) = C_2$ and $L_2 = L_{C_2}^{\text{sup}}$, it holds that

$$\begin{aligned} s_2 &= \underbrace{\alpha_2\cdots\alpha_2}_{C_2}\underbrace{\omega_2\cdots\omega_2}_{C_2}\underbrace{\alpha_2\cdots\alpha_2}_{C_2}\underbrace{\omega_2\cdots\omega_2}_{C_2}\underbrace{\alpha_2\cdots\alpha_2}_{C_1-C_2}\underbrace{\omega_2\cdots\omega_2}_{C_1-C_2} \\ &= \underbrace{\omega_1\cdots\omega_1}_{C_2}\underbrace{\omega_2\cdots\omega_2}_{C_2}\underbrace{\omega_1\cdots\omega_1}_{C_2}\underbrace{\omega_2\cdots\omega_2}_{C_2}\underbrace{\omega_1\cdots\omega_1}_{C_1-C_2}\underbrace{\omega_2\cdots\omega_2}_{C_1-C_2} \\ &\in L_2 \end{aligned}$$

and $c(s_2) = C_2$ since $C_1 - C_2 \leq C_2$ by assumption (if $C_1 - C_2 > C_2$, it is sufficient to add additional substrings $\underbrace{\omega_1\cdots\omega_1}_{C_2}\underbrace{\omega_2\cdots\omega_2}_{C_2}$). Since $\Sigma_1 \cap \Sigma_2 = \Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$, it further holds that,

$$\begin{aligned} s &= \underbrace{\alpha_1\cdots\alpha_1}_{C_2}\underbrace{\alpha_1\cdots\alpha_1}_{C_1}\underbrace{\omega_2\cdots\omega_2}_{C_2}\underbrace{\omega_2\cdots\omega_2}_{C_2}\underbrace{\omega_2\cdots\omega_2}_{C_1-C_2} \\ &= p(\underbrace{\alpha_1\cdots\alpha_1}_{C_2}\underbrace{\omega_1\cdots\omega_1}_{C_2}\underbrace{\alpha_1\cdots\alpha_1}_{C_1}\underbrace{\omega_2\cdots\omega_2}_{C_2}\underbrace{\omega_1\cdots\omega_1}_{C_2} \\ &\quad \underbrace{\omega_2\cdots\omega_2}_{C_2}\underbrace{\omega_1\cdots\omega_1}_{C_1-C_2}\underbrace{\omega_2\cdots\omega_2}_{C_1-C_2}) \\ &= p(s_1||s_2) \in p(L_1||L_2) = L \end{aligned}$$

and

$$\begin{aligned} c(\text{s}) &= |\text{p}^{\text{in}}(\underbrace{\alpha_1\ldots\alpha_1}_{C_2}\underbrace{\alpha_1\ldots\alpha_1}_{C_1})| - |\text{p}^{\text{out}}(\underbrace{\alpha_1\ldots\alpha_1}_{C_2}\underbrace{\alpha_1\ldots\alpha_1}_{C_1})| \\ &\quad C_1 + C_2 - 0 = C_1 + C_2. \end{aligned}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

In the second case, $C_1 < C_2$. In addition, for simplicity, we assume that $C_2 \leq 2\,C_1$. The extension for the case $C_2 > 2\,C_1$ is conceptually straightforward. Since the capacity of L_2 is $c(L_2) = C_2$ and $L_2 = L_{C_2}^{\sup}$, it holds for an arbitrary $\alpha_1 \in \Sigma_1^{\text{in}}$ and $\omega_1 = m_1(\alpha_1) \in \Sigma_1^{\text{out}}$ that

$$s_1 = \underbrace{\alpha_1 \cdots \alpha_1}_{C_1}\underbrace{\omega_1 \cdots \omega_1}_{C_1}\underbrace{\alpha_1 \cdots \alpha_1}_{C_2-C_1}\underbrace{\omega_1 \cdots \omega_1}_{C_2-C_1}\underbrace{\alpha_1 \cdots \alpha_1}_{C_1}\underbrace{\omega_1 \cdots \omega_1}_{C_1} \in L_1$$

and $c(s_1) = C_1$. Now consider that $\alpha_2 = \omega_1 \in \Sigma_2^{\text{in}} = \Sigma_1^{\text{out}}$ and $\omega_2 = m_2(\alpha_2) \in \Sigma_2^{\text{out}}$. Since the capacity of L_2 is $c(L_2) = C_2$ and $L_2 = L_{C_2}^{\sup}$, it holds that

$$\begin{aligned} s_2 &= \underbrace{\alpha_2 \cdots \alpha_2}_{C_1}\underbrace{\alpha_2 \cdots \alpha_2}_{C_2-C_1}\underbrace{\omega_2 \cdots \omega_2}_{C_2}\underbrace{\alpha_2 \cdots \alpha_2}_{C_1}\underbrace{\omega_2 \cdots \omega_2}_{C_1} \\ &= \underbrace{\omega_1 \cdots \omega_1}_{C_1}\underbrace{\omega_1 \cdots \omega_1}_{C_2-C_1}\underbrace{\omega_2 \cdots \omega_2}_{C_2}\underbrace{\omega_1 \cdots \omega_1}_{C_1}\underbrace{\omega_2 \cdots \omega_2}_{C_1} \in L_2 \end{aligned}$$

and $c(s_2) = C_2$. Since $\omega_1 \in \Sigma_1 \cap \Sigma_2 = \Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$, it follows that

$$\begin{aligned} s &= \underbrace{\alpha_1 \cdots \alpha_1}_{C_1}\underbrace{\alpha_1 \cdots \alpha_1}_{C_2-C_1}\underbrace{\alpha_1 \cdots \alpha_1}_{C_1}\underbrace{\omega_2 \cdots \omega_2}_{C_2}\underbrace{\omega_2 \cdots \omega_2}_{C_1} \\ &= p(\underbrace{\alpha_1 \cdots \alpha_1}_{C_1}\underbrace{\omega_1 \cdots \omega_1}_{C_1}\underbrace{\alpha_1 \cdots \alpha_1}_{C_2-C_1}\underbrace{\omega_1 \cdots \omega_1}_{C_2-C_1}\underbrace{\alpha_1 \cdots \alpha_1}_{C_1} \\ &\quad \underbrace{\omega_2 \cdots \omega_2}_{C_2}\underbrace{\omega_1 \cdots \omega_1}_{C_1}\underbrace{\omega_2 \cdots \omega_2}_{C_1}) \\ &\in p(s_1||s_2) \subseteq p(L_1||L_2) = L. \end{aligned}$$

and

$$\begin{aligned} c(\mathrm{s}) = |p^{\text{in}}(\underbrace{\alpha_1 \ldots \alpha_1}_{C_1}\underbrace{\alpha_1 \ldots \alpha_1}_{C_2-C_1}\underbrace{\alpha_1 \ldots \alpha_1}_{C_1})| - \\ |p^{\text{out}}(\underbrace{\alpha_1 \ldots \alpha_1}_{C_1}\underbrace{\alpha_1 \ldots \alpha_1}_{C_2-C_1}\underbrace{\alpha_1 \ldots \alpha_1}_{C_1})| = C_1 + C_2. \end{aligned}$$

That is, indeed, $c(L) = C = C_1 + C_2$ □

APPENDIX C PROOF OF LEMMA 3

We next prove Lemma 3. Again, we write $L_{C_1}^{\sup}$ and $L_{C_2}^{\sup}$ instead of $L_{C_1}^{\sup}(\Sigma_1^{\text{in}}, \Sigma_1^{\text{out}}, m_1)$ and $L_{C_2}^{\sup}(\Sigma_2^{\text{in}}, \Sigma_2^{\text{out}}, m_2)$.

Proof. We show that $f(s) = m_1^{-1}(f_2(s_2))\, f_1(s_1)$ by induction on the length of prefixes of $s_k = \text{pre}_k(s)$. For the initialization, we start from $k = 0$. It holds that $s_0 = \epsilon$ and $s_0 = p(s_{1,0}||s_{2,0})$ and $f(s_0) = m_1^{-1}(f_2(s_{2,0}))\, f_1(s_{1,0}) = \epsilon$ for $s_{1,0} = \epsilon \leq s_1$, $s_{2,0} = \epsilon \leq s_2$.

For the induction step, assume for some $k < |s|$, $s_{1,k} \leq s_1$, $s_{2,k} \leq s_2$ that $s_k = p(s_{1,k}||s_{2,k})$ and $f(s_k) = m_1^{-1}(f_2(s_{2,k}))\, f_1(s_{1,k})$. We have to show that for $s_{k+1} = s_k\sigma \leq s$, there are $s_{1,k+1} \leq s_1$, $s_{2,k+1} \leq s_2$ such that $s_{k+1} = p(s_{1,k+1}||s_{2,k+1})$ and $f(s_{k+1}) = m_1^{-1}(f_2(s_{2,k+1}))\, f_1(s_{1,k+1})$.

Consider $s_{k+1} = s_k\sigma \leq s$ and write $f(s_k) = \alpha_1 \cdots \alpha_j$ for some $0 \leq j \leq C$ with $\alpha_1, \ldots, \alpha_j \in \Sigma^{\text{in}}$ (with a slight abuse of notation, we define $f(s_k) = \epsilon$ if $j = 0$). Then, it is possible that (i) $\sigma \in \Sigma^{\text{in}}$ or (ii) $\sigma \in \Sigma^{\text{out}}$.

(i) Assume that $\sigma = \alpha_{j+1} \in \Sigma^{\text{in}}$. Then, it must be the case that $j < C$. Consider the contrary, that is, $j = C = C_1 + C_2$. Since $|f(s_k)| = |m_1^{-1}(f_2(s_{2,k}))f_1(s_{1,k})|$ and $c(s_{i,k}) \leq C_i$ for $i = 1, 2$, this implies that $|f_i(s_{i,k})| = C_i$. Since $s_{k+1} \in p(\overline{L}_1||\overline{L}_2)$, $s_k = p(s_{1,k}||s_{2,k})$ and $\alpha_{j+1} \notin \Sigma_2$, it must further be the case that $s_{k+1} = p(s_{1,k}u\alpha_{j+1}||s_{2,k}u)$ for some $u \in \Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$. Considering that $|f_2(s_{2,k})| = C_2$ it follows that $u = \epsilon$ (otherwise, $|f_2(s_{2,k}u)| > C_2$. But then, $|f_1(s_{1,k}u\alpha_{j+1})| = |f_1(s_{1,k}\alpha_{j+1})| = C_1 + 1$, which implies that $s_{1,k}\alpha_{j+1} \notin \overline{L}_1$ and hence $s_{k+1} \notin p(\overline{L}_1||\overline{L}_2)$. This contradicts the assumption that $s \in p(\overline{L}_1||\overline{L}_2)$. Hence, indeed $j < C$. Accordingly, $|f_1(s_{1,k})| < C_1$ or $|f_1(s_{1,k})| = C_1$ and $|f_2(s_{2,k})| < C_2$. If $|f_1(s_{1,k})| < C_1$, we know that $m_1^{-1}(f_2(s_{2,k})) = \alpha_1 \cdots \alpha_l$ and $f_1(s_{1,k}) = \alpha_{l+1} \cdots \alpha_j$ for $l \leq C_2$ and $j - l < C_1$. Then, it holds that $s_{1,k}\alpha_{j+1} \in \overline{L}_1$ since $L_1 = L_{C_1}^{\sup}$. Furthermore, $s_{k+1} = s_k\alpha_{j+1} = p(s_{1,k+1}||s_{2,k+1})$ for $s_{1,k+1} = s_{1,k}\alpha_{j+1}$ and $s_{2,k+1} = s_{2,k}$ and $f(s_{k+1}) = \alpha_1 \cdots \alpha_l\alpha_{l+1} \cdots \alpha_j\alpha_{j+1} = m_2^{-1}(f_2(s_{2,k+1}))f_1(s_{1,k+1})$. If $|f_1(s_{1,k})| = C_1$ and $|f_2(s_{2,k})| < C_2$, we have that $m_1^{-1}(f_2(s_{2,k})) = \alpha_1 \cdots \alpha_l$ and $f_1(s_{1,k}) = \alpha_{l+1} \cdots \alpha_j$ for $l < C_2$ and $j - l = C_1$. Consider $\gamma_{l+1} = m_1(\alpha_{l+1})$. Since $L_1 = L_{C_1}^{\sup}$, $L_2 = L_{C_2}$ and $\gamma_{l+1} \in \Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$, it must be the case that $s_{1,k}\gamma_{l+1} \in \overline{L}_1$ and $s_{2,k}\gamma_{l+1} \in \overline{L}_2$. But then, also $s_{1,k}\gamma_{l+1}\alpha_{j+1} \in \overline{L}_1$ since $|f_1(s_{1,k}\gamma_{l+1})| = C_1 - 1 < C_1$. That is, defining $s_{1,k+1} = s_{1,k}\gamma_{l+1}\alpha_{j+1}$ and $s_{2,k+1} = s_{2,k}\gamma_{l+1}$, we have that $s_{k+1} = s_k\alpha_{j+1} = p(s_{1,k+1}||s_{2,k+1})$ since $\alpha_{j+1} \notin \Sigma_2$. In addition, it holds that $f(s_{k+1}) = \alpha_1 \cdots \alpha_j\alpha_{j+1}$, $f_1(s_{1,k+1}) = \alpha_{l+2} \cdots \alpha_{j+1}$ and $m_1^{-1}(f_2(s_{2,k+1}) = \alpha_1 \cdots \alpha_l\alpha_{l+1}$ with $|f_1(s_{1,k+1})| = j + 1 - (l + 1) = j - l = C_1$ and $|f_2(s_{2,k+1})| = l + 1 \leq C_2$. Moreover, $f(s_{k+1}) = \alpha_1 \cdots \alpha_l\alpha_{l+1} \cdots \alpha_j\alpha_{j+1} = m_2^{-1}(f_2(s_{2,k+1}))f_1(s_{1,k+1})$.

(ii) If $\sigma \in \Sigma^{\text{out}}$, it must be the case that $j > 0$ (if $j = 0$, $|f(s_k\sigma)| = -1 < 0$). Accordingly, $|f_2(s_{2,k})| > 0$ or $|f_2(s_{2,k})| = 0$ and $|f_1(s_{1,k})| > 0$. If $|f_2(s_{2,k})| > 0$, we know that $f_1(s_{1,k}) = \alpha_{l+1} \cdots \alpha_j$ and $m_1^{-1}(f_2(s_{2,k})) = \alpha_1 \cdots \alpha_l$ for some $l > 0$. Then, it holds that $s_{2,k}\omega_1 \in \overline{L}_2$ since $L_2 = L_{C_2}^{\sup}$ and L_2 is OP. That is, $\sigma = \omega_1$. Furthermore, $s_{k+1} = s_k\omega_1 = p(s_{1,k+1}||s_{2,k+1})$ for $s_{1,k+1} = s_k$ and $s_{2,k+1} = s_{2,k}\omega_1$ and $f(s_{k+1}) = \alpha_2 \cdots \alpha_j = m_1^{-1}(f_2(s_{2,k+1}))f_1(s_{1,k+1})$. If $|f_2(s_{2,k})| = 0$ and $|f_1(s_{1,k})| > 0$, we have that $f_1(s_{1,k}) = \alpha_1 \cdots \alpha_j$ and $m_1^{-1}(f_2(s_{2,k})) = \epsilon$. Since $L_1 = L_{C_1}^{\sup}$, it holds that $s_{1,k}\gamma_1 \in \overline{L}_1$ for $\gamma_1 = m_1(\alpha_1)$. Since $L_2 = L_{C_2}^{\sup}$, $\gamma_1 \in \Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$ and $\omega_1 = m_2(\gamma_1) = m(\alpha_1) \in \Sigma_2^{\text{out}}$, it is further the case that $s_{2,k}\gamma_1\omega_1 \in \overline{L}_2$. That is, $s_{k+1} = s_k\omega_1 = p(s_{1,k+1}||s_{2,k+1})$ for $s_{1,k+1} = s_{1,k}\gamma_1$ and $s_{2,k+1} = s_{2,k}\gamma_1\omega_1$. In addition, it holds that $f(s_{k+1}) = \alpha_2 \cdots \alpha_j = m_2^{-1}(f_2(s_{2,k+1}))f_1(s_{1,k+1})$ since $m_2^{-1}(f_2(s_{2,k+1})) = m_2^{-1}(f_2(s_{2,k}\gamma_1\omega_1)) = m_2^{-1}(f_2(s_{2,k})) = \epsilon$ and $f_1(s_{1,k+1}) = f_1(s_{1,k}\gamma_1) = \alpha_2 \cdots \alpha_j$.

Since $k < |s|$ was chosen arbitrary, the proven relation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3010030, IEEE Access

also holds for $k = |s| - 1$. That is,

$$f(s) = f(s_{k+1}) = m_1^{-1}(f_2(s_{2,k+1}))f_1(s_{1,k+1}) = m_1^{-1}(f_2(s_2))f_1(s_2).$$

□

APPENDIX D PROOF OF LEMMA 4

We next prove Lemma 4. As before, we write $L_{C_1}^{\sup}$ and $L_{C_2}^{\sup}$ instead of $L_{C_1}^{\sup}(\Sigma_1^{\text{in}}, \Sigma_1^{\text{out}}, m_1)$ and $L_{C_2}^{\sup}(\Sigma_2^{\text{in}}, \Sigma_2^{\text{out}}, m_2)$.

Proof. Consider an arbitrary string $t \in \overline{L_1||L_2}$ and let $\sigma \in \Sigma$ such that $p(t)\sigma \in p(\overline{L_1||L_2})$. We write $s = p(t)$. According to Lemma 3, it holds that $s = p(s_1||s_2)$ for $s_1 \in \overline{L}_1$ and $s_2 \in \overline{L}_2$. In addition, $f(s) = m_1^{-1}(f_2(s_2))f_1(s_1) = \alpha_1 \cdots \alpha_l \alpha_{l+1} \cdots \alpha_j$ with $|m_1^{-1}(f_2(s_2))| = |\alpha_1 \cdots \alpha_l| = l$ and $|f_1(s_1)| = |f_1(\alpha_{l+1} \cdots \alpha_j)| = j - l$ and $0 \leq j \leq C = C_1 + C_2$.

We now consider that (i) $\sigma \in \Sigma^{\text{in}}$ or (ii) $\sigma \in \Sigma^{\text{out}}$.

(i) If $\sigma \in \Sigma^{\text{in}} = \Sigma_1^{\text{in}}$, it must be the case that $j < C$. Otherwise, $|f(s\sigma)| = |f(s)\sigma| = j + 1 > C$. There are two cases. In the first case, $|f_1(s_1)| < C_1$. Since $L_1 = L_{C_1}^{\sup}$, it holds that $s_1\sigma \in \overline{L}_1$. Since $\Sigma_1^{\text{in}} \cap \Sigma_2 = \emptyset$, it follows that $t\sigma \in s_1\sigma||s_2 \subseteq \overline{L}_1||\overline{L}_2$ and $p(t\sigma) = p(t)\sigma = s\sigma$. That is, (14) is fulfilled with $v = \sigma \in (\Sigma_1 \cup \Sigma_2)^\star$. In the second case, $|f_1(s_1)| = C_1$ and hence $|f_2(s_2)| < C_2$. We write $\gamma_{l+1} = m_1(\alpha_{l+1}) \in \Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$. Since $L_1 = L_{C_1}^{\sup}$ and $L_2 = L_{C_2}^{\sup}$, it is the case that $s_1\gamma_{l+1} \in \overline{L}_1$ and $s_2\gamma_{l+1} \in \overline{L}_2$. Furthermore, $\gamma_{l+1} \in \Sigma_1^{\text{out}}$, implies $|f_1(s_1\gamma_{l+1})| = |\alpha_{l+2} \cdots \alpha_j| = C_1 - 1 < C_1$. That is, $s_1\gamma_{l+1}\sigma \in \overline{L}_1$ since $L_1 = L_{C_1}^{\sup}$. Together, we have that $t\gamma_{l+1}\sigma \in s_1\gamma_{l+1}\sigma||s_2\gamma_{l+1} \subseteq \overline{L}_1||\overline{L}_2$ and $p(t\gamma_{l+1}\sigma) = p(t)\sigma = s\sigma$. That is, (14) is fulfilled with $v = \gamma_{l+1}\sigma$.

(ii) if $\sigma \in \Sigma^{\text{out}} = \Sigma_2^{\text{out}}$, it must be the case that $j > 0$. Otherwise, $|f(s\sigma)| = j - 1 < 0$. There are two cases. In the first case, $|f_2(s_2)| > 0$. Since $L_2 = L_{C_2}^{\sup}$, it holds that $s_2\sigma \in \overline{L}_2$. Since $\Sigma_2^{\text{in}} \cap \Sigma_1 = \emptyset$, it follows that $t\sigma \in s_1||s_2\sigma \subseteq \overline{L}_1||\overline{L}_2$ and $p(t\sigma) = p(t)\sigma = s\sigma$. That is, (14) is fulfilled with $v = \sigma \in (\Sigma_1 \cup \Sigma_2)^\star$. In the second case, $|f_2(s_2)| = 0$ and hence $|f_1(s_1)| > 0$. In addition, it must hold that $\sigma = \omega_1 = m(\alpha_1)$ since $f(s) = \alpha_1 \cdots \alpha_j$. We write $\gamma_1 = m_1(\alpha_l) \in \Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$. Since $L_1 = L_{C_1}^{\sup}$ and $L_2 = L_{C_2}^{\sup}$, it is the case that $s_1\gamma_1 \in \overline{L}_1$ and $s_2\gamma_1 \in \overline{L}_2$. Furthermore, $\gamma_1 \in \Sigma_2^{\text{in}}$, implies $|m_1^{-1}(f_2(s_2\gamma_1))| = |\alpha_1| = 1 > 0$. That is, $s_1\gamma_1\omega_1 \in \overline{L}_1$ since $m_2(\gamma_1) = m(\alpha_1) = \omega_1$ and $L_2 = L_{C_2}^{\sup}$. Together, we have that $t\gamma_1\sigma = t\gamma_1\omega_1 \in s_1\gamma_{l+1}||s_2\gamma_{l+1}\omega_1 \subseteq \overline{L}_1||\overline{L}_2$ and $p(t\gamma_{l+1}\sigma) = p(t)\sigma = s\sigma$. That is, (14) is fulfilled with $v = \gamma_1\sigma$.

In summary, (14) holds in all possible cases, which proves the lemma. □

APPENDIX E PROOF OF COROLLARY 2

We next prove Corollary 2. To this end, we first state the following result from the existing literature [50], [61].

Lemma 5. *Consider alphabets Σ_1, Σ_2 and write $\Sigma_\cap = \Sigma_1 \cap \Sigma_2$, $\Lambda = \Sigma_1 \cup \Sigma_2$. Assume that $\Sigma_\cap \subseteq \Pi \subseteq \Lambda$ for some alphabet Π and define the projections $p_i : \Sigma_i^\star \to (\Sigma_i \cap \Pi)^\star$ for $i = 1, 2$ and $p : \Lambda^\star \to \Pi^\star$. Then, it holds for any $L_1 \subseteq \Sigma_1^\star$ and $L_2 \subseteq \Sigma_2^\star$ that*

$$p(L_1||L_2) = p_1(L_1)||p_2(L_2). \tag{15}$$

We next provide the proof of Corollary 2.

Proof. We use the same notation as in the proof or Corollary 1 with $\Gamma_k = \Sigma_1^{\text{in}} \cup \Sigma_k^{\text{out}}$, $\Lambda_k = \bigcup_{i=1}^k \Sigma_i$, $\Pi_k = \Sigma_1^{\text{in}} \cup \Sigma_k$ for $k = 2, \ldots, n$, $p_{\Lambda_k,\Gamma_k} : \Lambda_k^\star \to \Gamma_k^\star$, $p_{\Pi_k,\Gamma_k} : \Pi_k^\star \to \Gamma_k^\star$ and the languages $L_{1,k} = p_{\Lambda_k,\Gamma_k}(L_1|| \cdots ||L_k)$ for $k = 2, \ldots, n$.

Next, we prove the assertion in Corollary 2 by induction. For the initialization, we know that p_{Λ_2,Γ_2} is a natural observer for $L_1||L_2$ from Theorem 6 and $L_{1,2} = L_{C_1+C_2}^{\sup}$ from Theorem 4.

For the induction step, we assume that p_{Λ_k,Γ_k} is a natural observer for $L_1|| \cdots ||L_k$. Then, we show that also $p_{\Lambda_{k+1},\Gamma_{k+1}}$ is a natural observer for $L_1|| \cdots ||L_k||L_{k+1}$. To this end, we first note that $\Lambda_k = \Sigma_1 \cup \cdots \cup \Sigma_k = \Sigma_1^{\text{in}} \cup \Sigma_1^{\text{out}} \cup \Sigma_2 \cdots \Sigma_k^{\text{in}} \cup \Sigma_k^{\text{out}}$ and $\Sigma_{k+1} = \Sigma_{k+1}^{\text{in}} \cup \Sigma_{k+1}^{\text{out}}$ such that $\Lambda_k \cap \Sigma_{k+1} = \Sigma_k^{\text{out}} = \Sigma_{k+1}^{\text{in}}$. Then, we compute

$$\begin{aligned} p_{\Lambda_{k+1},\Gamma_{k+1}}(L_1|| \cdots ||L_{k+1}) &= \\ &= p_{\Pi_{k+1},\Gamma_{k+1}}\big(p_{\Lambda_{k+1},\Pi_{k+1}}(L_1|| \cdots ||L_k||L_{k+1})\big) \\ &= p_{\Pi_{k+1},\Gamma_{k+1}}\big(p_{\Lambda_k,\Gamma_k}(L_1|| \cdots ||L_k)||L_{k+1}\big) \end{aligned}$$

because of Lemma 5. Here, we already know that p_{Λ_k,Γ_k} is a natural observer for $L_1|| \cdots ||L_k$ due to the induction assumption. In order to show that $p_{\Lambda_{k+1},\Gamma_{k+1}}$ is a natural observer for $L_1|| \cdots ||L_{k+1}$, we take an arbitrary string $t \in L_1|| \cdots ||L_{k+1}$ and $u \in \Gamma_{k+1}^\star$ and assume that $p_{\Lambda_{k+1},\Gamma_{k+1}}(t)u \in p_{\Lambda_{k+1},\Gamma_{k+1}}(L_1|| \cdots ||L_{k+1})$. We first observe that

$$\begin{aligned} p_{\Lambda_{k+1},\Gamma_{k+1}}(t)u = p_{\Pi_{k+1},\Gamma_{k+1}}p_{\Lambda_{k+1},\Pi_{k+1}}(t)u \in \\ \in p_{\Pi_{k+1},\Gamma_{k+1}}p_{\Lambda_{k+1},\Pi_{k+1}}(L_1|| \cdots ||L_{k+1}) \end{aligned}$$

That is, there must be a $w \in \Pi_{k+1}^\star$ such that $p_{\Pi_{k+1},\Gamma_{k+1}}(w) = u$ and

$$\begin{aligned} p_{\Lambda_{k+1},\Pi_{k+1}}(t)w &\in p_{\Lambda_{k+1},\Pi_{k+1}}(L_1|| \cdots ||L_{k+1}) \\ &= p_{\Lambda_k,\Gamma_k}(L_1|| \cdots ||L_k)||L_{k+1}. \end{aligned}$$

Here, we know that $L_{k+1} = L_{C_{k+1}}^{\sup}$ by assumption and $p_{\Lambda_k,\Gamma_k}(L_1|| \cdots ||L_k) = L_{C_1+\cdots+C_k}^{\sup}$ because of Theorem 1. Then, Theorem 6 implies that $p_{\Lambda_{k+1},\Pi_{k+1}}$ is a natural observer for $L_1|| \cdots ||L_{k+1}$. Accordingly, there is a $v \in \Lambda_{k+1}^\star$ such that $p_{\Lambda_{k+1},\Pi_{k+1}}(v) = w$ and $tw \in L_1|| \cdots ||L_{k+1}$. Recalling that $p_{\Pi_{k+1},\Gamma_{k+1}}(w) = u$, we found $v \in \Lambda_{k+1}^\star$ such that $p_{\Lambda_{k+1},\Gamma_{k+1}}(v) = u$ and $tv \in L_1|| \cdots ||L_{k+1}$. Hence, $p_{\Lambda_{k+1},\Gamma_{k+1}}$ is indeed a natural observer.

In particular, the obtained result is true for $k + 1 = n$. That is, $p = p_{\Lambda_n,\Gamma_n}$ in Corollary 2 is a natural observer for $L_1|| \cdots ||L_n$. □

REFERENCES

[1] A. A. Desrochers, Modeling and control of automated manufacturing systems. IEEE Computer Society Press Washington, DC, 1989.

[2] P. M. Swamidass, Automated Manufacturing System. Encyclopedia of Production and Manufacturing Management. Springer, Boston, MA, 2000.

[3] D. Chao, "Automated manufacturing system: virtual-nets or non-virtual-nets?," IET Control Theory & Applications, vol. 3, no. 6, pp. 671–680, 2009.

[4] H. Hu and M. Zhou, "A petri net-based discrete-event control of automated manufacturing systems with assembly operations," IEEE Transactions on Control Systems Technology, vol. 23, no. 2, pp. 513–524, 2015.

[5] N. Du and H. Hu, "A robust prevention method for automated manufacturing systems with unreliable resources using petri nets," IEEE Access, vol. 6, pp. 78598–78608, 2018.

[6] S. Lafortune, "Discrete event systems: Modeling, observation, and control," Annual Review of Control, Robotics, and Autonomous Systems, vol. 2, pp. 141–159, 2019.

[7] B. S. Kim, Y. Jin, and S. Nam, "An integrative user-level customized modeling and simulation environment for smart manufacturing," IEEE Access, vol. 7, pp. 186637–186645, 2019.

[8] X. Guo, S. Wang, D. You, Z. Li, and X. Jiang, "A siphon-based deadlock prevention strategy for s 3 pr," IEEE Access, vol. 7, pp. 86863–86873, 2019.

[9] M. H. de Queiroz, J. E. Cury, and W. M. Wonham, "Multitasking supervisory control of discrete-event systems," Discrete Event Dynamic Systems, vol. 15, no. 4, pp. 375–395, 2005.

[10] W. Chao, Y. Gan, W. Wonham, and Z. Wang, "Nonblocking supervisory control of flexible manufacturing systems based on state tree structures," in Formal Methods in Manufacturing Systems: Recent Advances, pp. 1–19, IGI Global, 2013.

[11] P. N. Pena, T. A. Costa, R. S. Silva, and R. H. Takahashi, "Control of flexible manufacturing systems under model uncertainty using supervisory control theory and evolutionary computation schedule synthesis," Information Sciences, vol. 329, pp. 491–502, 2016.

[12] T. Sprock, C. Bock, and L. F. McGinnis, "Survey and classification of operational control problems in discrete event logistics systems (DELS)," International journal of production research, vol. 57, no. 15-16, pp. 5215–5238, 2019.

[13] R. El-Khalil and Z. Darwish, "Flexible manufacturing systems performance in us automotive manufacturing plants: a case study," Production Planning & Control, vol. 30, no. 1, pp. 48–59, 2019.

[14] K. Schmidt, T. Moor, and S. Perk, "Nonblocking hierarchical control of decentralized discrete event systems," IEEE Transactions on Automatic Control, vol. 53, no. 10, pp. 2252–2265, 2008.

[15] L. Feng and W. M. Wonham, "Supervisory control architecture for discrete-event systems," IEEE Transactions on Automatic Control, vol. 53, no. 6, pp. 1449–1461, 2008.

[16] J. E. Cury, M. H. de Queiroz, G. Bouzon, and M. Teixeira, "Supervisory control of discrete event systems with distinguishers," Automatica, vol. 56, pp. 93–104, 2015.

[17] K. Andersson, B. Lennartson, and M. Fabian, "Restarting manufacturing systems; restart states and restartability," IEEE Transactions on Automation Science and Engineering, vol. 7, pp. 486–499, July 2010.

[18] J. Zhang, G. Frey, A. Al-Ahmari, T. Qu, N. Wu, and Z. Li, "Analysis and control of dynamic reconfiguration processes of manufacturing systems," IEEE Access, vol. 6, pp. 28028–28040, 2018.

[19] Y. Koren, X. Gu, and W. Guo, "Reconfigurable manufacturing systems: Principles, design, and future trends," Frontiers of Mechanical Engineering, vol. 13, no. 2, pp. 121–136, 2018.

[20] M. Khalgui, O. Mosbahi, and Z. Li, "On reconfiguration theory of discrete-event systems: From initial specification until final deployment," IEEE Access, vol. 7, pp. 18219–18233, 2019.

[21] 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.

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

[23] Z. Li, M. Zhou, and N. Wu, "A survey and comparison of petri net-based deadlock prevention policies for flexible manufacturing systems," IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 38, no. 2, pp. 173–188, 2008.

[24] Y. Chen, Z. Li, and M. Zhou, "Behaviorally optimal and structurally simple liveness-enforcing supervisors of flexible manufacturing systems," IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 42, no. 3, pp. 615–629, 2012.

[25] M. Zhao, M. Uzam, and Y. Hou, "Near-optimal supervisory control of flexible manufacturing systems using divide-and-conquer iterative method," Advances in Mechanical Engineering, vol. 8, no. 3, p. 1687814016639823, 2016.

[26] Y. Hou and K. Barkaoui, "Deadlock analysis and control based on petri nets: A siphon approach review," Advances in Mechanical Engineering, vol. 9, no. 5, p. 1687814017693542, 2017.

[27] Y. Li, L. Yin, Y. Chen, Z. Yu, and N. Wu, "Optimal petri net supervisor synthesis for forbidden state problems using marking mask," Information Sciences, vol. 505, pp. 183–197, 2019.

[28] Y. Feng, K. Xing, M. Zhou, and H. Liu, "Liveness analysis and deadlock control for automated manufacturing systems with multiple resource requirements," IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 50, pp. 525–538, Feb 2020.

[29] H. Hu and Y. Liu, "Supervisor simplification for AMS based on petri nets and inequality analysis," IEEE Transactions on Automation Science and Engineering, vol. 11, no. 1, pp. 66–77, 2014.

[30] H. Hu, Y. Liu, and L. Yuan, "Supervisor simplification in FMSs: Comparative studies and new results using petri nets," IEEE Transactions on Control Systems Technology, vol. 24, no. 1, pp. 81–95, 2016.

[31] J. Luo and M. Zhou, "Petri-net controller synthesis for partially controllable and observable discrete event systems," IEEE Transactions on Automatic Control, vol. 62, no. 3, pp. 1301–1313, 2017.

[32] C. Chen and H. Hu, "Static and dynamic partitions of inequalities: A unified methodology for supervisor simplification," IEEE Transactions on Automatic Control, vol. 64, no. 11, pp. 4748–4755, 2019.

[33] 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.

[34] K. Schmidt and C. Breindl, "Maximally permissive hierarchical control of decentralized discrete event systems," IEEE Transactions on Automatic Control, vol. 56, no. 4, pp. 723–737, 2010.

[35] K. Cai and W. M. Wonham, "Supervisor localization: a top-down approach to distributed control of discrete-event systems," IEEE Transactions on Automatic Control, vol. 55, no. 3, pp. 605–618, 2010.

[36] M. H. De Queiroz and J. E. Cury, "Modular supervisory control of large scale discrete event systems," in Discrete Event Systems, pp. 103–110, Springer, 2000.

[37] R. J. Leduc, M. Lawford, and W. M. Wonham, "Hierarchical interface-based supervisory control-part ii: parallel case," IEEE Transactions on Automatic Control, vol. 50, no. 9, pp. 1336–1348, 2005.

[38] J. Zaytoon and B. Riera, "Synthesis and implementation of logic controllers–a review," Annual reviews in control, vol. 43, pp. 152–168, 2017.

[39] S. Kumar, R. Devaraj, and A. Sarkar, "A hybrid offline-online approach to adaptive downlink resource allocation over LTE," IEEE/CAA Journal of Automatica Sinica, vol. 6, no. 3, pp. 766–777, 2019.

[40] M. Goorden, J. van de Mortel-Fronczak, M. Reniers, W. Fokkink, and J. Rooda, "Modeling guidelines for component-based supervisory control synthesis," in International Conference on Formal Aspects of Component Software, pp. 3–24, Springer, 2019.

[41] L. Grigorov, B. E. Butler, J. E. Cury, and K. Rudie, "Conceptual design of discrete-event systems using templates," Discrete Event Dynamic Systems, vol. 21, no. 2, pp. 257–303, 2011.

[42] M. Teixeira, J. E. Cury, and M. H. de Queiroz, "Exploiting distinguishers in local modular control of discrete-event systems," IEEE Transactions on Automation Science and Engineering, vol. 15, no. 3, pp. 1431–1437, 2018.

[43] D. Côté and R. St-Denis, "Component-based method for the modeling and control of modular production systems," IEEE Transactions on Control Systems Technology, vol. 21, no. 5, pp. 1570–1585, 2012.

[44] R. A. Williams, B. Benhabib, and K. Smith, "A hybrid supervisory control system for flexible manufacturing workcells," in Proceedings of the 1994 IEEE International Conference on Robotics and Automation, pp. 2551–2556, IEEE, 1994.

[45] A. Nooruldeen and K. W. Schmidt, "State attraction under language specification for the reconfiguration of discrete event systems," IEEE Transactions on Automatic Control, vol. 60, no. 6, pp. 1630–1634, 2015.

[46] K. W. Schmidt, "Reconfigurability of behavioural specifications for manufacturing systems," International Journal of Control, vol. 90, no. 12, pp. 2605–2617, 2017.

[47] T. Jiao, Y. Gan, G. Xiao, and W. Wonham, "Exploiting symmetry of discrete-event systems by relabeling and reconfiguration," IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2018.
[48] A. Nooruldeen and K. W. Schmidt, "Order-preserving models for the supervisory control of flexible manufacturing systems," Çankaya University Journal of Science and Engineering, vol. 16, no. 2, pp. 70–86, 2019.
[49] F. Lin, W. Wang, L. Han, and B. Shen, "State estimation of multi-channel networked discrete event systems," IEEE Transactions on Control of Network Systems, 2019.
[50] W. Wonham, Supervisory Control of Discrete Event Systems. Systems Control Group, University of Toronto, Canada, 2010.
[51] K. C. Wong and W. M. Wonham, "Hierarchical control of discrete-event systems," Discrete Event Dynamic Systems: Theory and Applications, vol. 6, no. 3, pp. 241–273, 1996.
[52] S.-J. Park and J.-T. Lim, "Robust and nonblocking supervisory control of nondeterministic discrete event systems using trajectory models," IEEE Transactions on Automatic Control, vol. 47, no. 4, pp. 655–658, 2002.
[53] R. Malik and R. Leduc, "Hierarchical interface-based supervisory control using the conflict preorder," in 11th International Workshop on Discrete Event Systems, vol. 45, pp. 163–168, The international Federation of Automatic Control, 2012.
[54] H. Hu, M. Zhou, Z. Li, and Y. Tang, "Deadlock-free control of automated manufacturing systems with flexible routes and assembly operations using petri nets," IEEE Transactions on Industrial Informatics, vol. 9, no. 1, pp. 109–121, 2013.
[55] R. Su, J. H. van Schuppen, and J. E. Rooda, "Aggregative synthesis of distributed supervisors based on automaton abstraction," IEEE Transactions on Automatic Control, vol. 55, no. 7, pp. 1627–1640, 2010.
[56] J. Huang and R. Kumar, "Optimal nonblocking directed control of discrete event systems," IEEE Transactions on Automatic Control, vol. 53, no. 7, pp. 1592–1603, 2008.
[57] H. Hu and Y. Liu, "Supervisor synthesis and performance improvement for automated manufacturing systems by using petri nets," IEEE Transactions on Industrial Informatics, vol. 11, no. 2, pp. 450–458, 2015.
[58] S. Ware and R. Su, "Time optimal synthesis based upon sequential abstraction and its application to cluster tools," IEEE Transactions on Automation Science and Engineering, vol. 14, no. 2, pp. 772–784, 2017.
[59] F. Hagebring and B. Lennartson, "Time-optimal control of large-scale systems of systems using compositional optimization," Discrete Event Dynamic Systems, vol. 29, no. 3, pp. 411–443, 2019.
[60] M. Silva, Modeling, Analysis and Control of Discrete Event Systems as Petri Nets. Encyclopedia of Systems and Control, Springer, London, 2020.
[61] K. Schmidt, J. Reger, and T. Moor, "Hierarchical control for structural decentralized DES," Discrete event systems, p. 279, 2004.

ANAS NOORULDEEN received the B.S. degree in Electronic and Control Engineering Techniques from the College of Technology, Kirkuk, Iraq, in 2010, the M.S. and Ph.D. degrees in Electronic and Communication Engineering from Çankaya University, Ankara, Turkey, in 2012 and 2020, respectively. From 2013 to 2015, he was a Research Assistant with the Department of Electrical Engineering of Kirkuk University, Iraq. His research interests include supervisory control of discrete event systems, automata, order-preserving models, flexible and reconfigurable manufacturing systems.

Dr. Nooruldeen was awarded an M.S. research grant and a full Ph.D. scholarship supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) in December 2011 and February 2016, respectively.

KLAUS WERNER SCHMIDT received the Diploma and Ph.D. degrees from the University of Erlangen-Nürnberg, Germany, in 2002 and 2005, respectively, both in Electrical, Electronic, and Communication Engineering. He is currently a Professor at the Department of Electrical & Electronics Engineering, Middle East Technical University, Ankara.

His research interests include supervisory control for discrete event systems, industrial automation systems, industrial communication networks, intelligent transportation systems and industrial project control. He is an Associate Editor for Discrete Event Dynamic Systems and the Turkish Journal of Electrical Engineering & Computer Sciences.