



**ORDER-PRESERVING MODELS FOR DISCRETE EVENT SYSTEMS:
THEORY AND APPLICATIONS**

ANAS NOORULDEEN

FEBRUARY 2020

ORDER-PRESERVING MODELS FOR DISCRETE EVENT SYSTEMS:
THEORY AND APPLICATIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES OF
ÇANKAYA UNIVERSITY

BY
ANAS NOORULDEEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF
DOCTOR OF PHILOSOPHY
IN
THE DEPARTMENT OF
ELECTRONIC AND COMMUNICATION ENGINEERING

FEBRUARY 2020

Title of the Thesis: Order-preserving Models for Discrete Event Systems: Theory and Applications.

Submitted by **Anas NOORULDEEN**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University.



Prof. Dr. Can ÇOGUN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy.

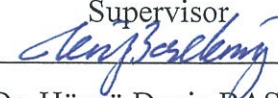


Prof. Dr. Sıtkı Kemal İDER
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Doctor of Philosophy.



Prof. Dr. Klaus Werner SCHMIDT
Supervisor



Assoc. Prof. Dr. Hüsnü Deniz BAŞDEMİR
Co-Supervisor

Examination Date: 04.02.2020

Examining Committee Members

Assoc. Prof. Dr. Afşar SARANLI

(METU)

Prof. Dr. Klaus Werner SCHMIDT

(METU)

Asst. Prof. Dr. Ulaş BELDEK

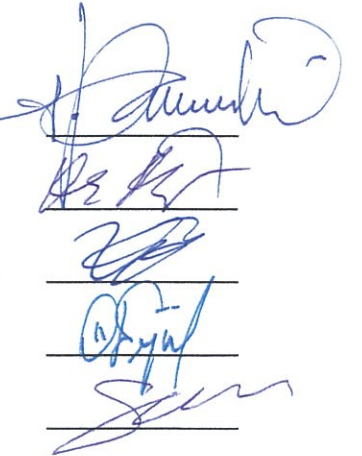
(Çankaya Univ.)

Asst. Prof. Dr. Özgür ERGÜL

(Atılım Univ.)

Asst. Prof. Dr. Selma ÖZAYDIN

(Çankaya Univ.)



STATEMENT OF NON-PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: Anas NOORULDEEN

Signature :

A handwritten signature in blue ink, consisting of a large, stylized 'A' followed by a horizontal line.

Date :

17.02.2020

ABSTRACT

ORDER-PRESERVING MODELS FOR DISCRETE EVENT SYSTEMS: THEORY AND APPLICATIONS

NOORULDEEN, Anas

Ph.D., Department of Electronic and Communication Engineering

Supervisor: Prof. Dr. Klaus Werner SCHMIDT

Co-Supervisor: Assoc. Prof. Dr. Hüsnü Deniz BAŞDEMİR

February 2020, 80 pages

Flexible manufacturing systems (FMS) are characterized by the processing of different product types on the same manufacturing systems. In particular, it is possible that the paths of different product types through an FMS overlap and different product types share the same production components such as machines or conveyor belts. That is, when designing controllers for FMS, it is required to keep track of products traveling through the FMS in order to process products correctly. In particular, it is important that the sequential order of different products is captured by dynamic models of FMS. In this context, the modeling formalism of discrete event systems (DES) is suitable since it allows capturing the sequential behavior of FMS.

Accordingly, this thesis develops a new modeling technique for the supervisory control of FMS in the framework of DES. In particular, the thesis considers the general case of an FMS, where different product types can share production components and production components can hold multiple products. It is first pointed out that a suitable model for such production component needs to keep track of the product type and the order of products entering and leaving production components. Then, order-preserving languages are introduced as a new model for FMS. Several important properties of such order-preserving languages are formally proved and their benefit for modeling FMS is discussed.

In addition, a general method for algorithmically constructing the required order-preserving models using finite state automata is proposed. The practicability of the developed method is demonstrated by several application examples.

Keywords: Discrete Event Systems, Flexible Manufacturing Systems, Supervisory Control, Order-Preserving Models.

ÖZ

AYRIK OLAYLI SİSTEMLER İÇİN SIRASAL DEVAMLILIĞI KORUMA MODELLERİ: KURAM VE UYGULAMALAR

NOORULDEEN, Anas

Doktora, Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Tez Yöneticisi: Prof. Dr. Klaus Werner SCHMIDT

Ortak Tez Yöneticisi : Assoc. Prof. Dr. Hüsnü Deniz BAŞDEMİR

Şubat 2020, 80 sayfa

Esnek Üretim Sistemleri (FMS), farklı ürün türlerinin aynı imalat sistemleri üzerinde işlenmesi ile karakterizedir. Bir FMS çakışması yoluyla farklı ürün türlerine ait yolların ve farklı ürün türlerinin, makine veya taşıma bantları gibi aynı üretim bileşenlerini paylaşmaları mümkündür. Yani, FMS ile ilgili kontrol birimlerini tasarlariken ürünleri doğru bir şekilde işlemek için FMS aracılığıyla taşınan ürünleri takip etmek gereklidir. Farklı ürünlerin sıralı düzeninin FMS'nin dinamik modelleri tarafından yakalanması özellikle önem arz etmektedir. Bu bağlamda ayrik olaylı sistemlerin (DES) modelleme formalizmi FMS'nin sıralı davranışını yakalamaya olanak sağladığı için uygundur.

Dolayısıyla bu tezde, DES çerçevesinde FMS'nin denetleyici kontrolüne yönelik yeni bir modelleme tekniği geliştirilmiştir. Tez bilhassa farklı ürün türlerinin üretim bileşenlerini paylaşabileceği ve bu üretim bileşenlerinin ise çoklu ürünleri barındırabileceği FMS'nin genel bir durumunu ele almaktadır. İlk olarak söz konusu üretim bileşeni için uygun bir modelin, bu ürünün türünü ve üretim bileşenlerine giren ve çıkan ürünlerin sırasını takip etmesi gerektiğine dikkat çekilmiştir. Sonrasında sırasal devamlılığı koruma dilleri, FMS için yeni bir model olarak sunulmuştur. Bu sırasal devamlılığı koruma dillerinin birçok önemli özelliği, usulen ispat edilmiş ve FMS modellemesi için faydaları tartışılmıştır.

Bununla birlikte, sonlu durum otamatı kullanılarak gerekli sırasal devamlılığı koruma modellerini algoritmik olarak oluşturmaya ilişkin genel bir yöntem önerilmiştir. Geliştirilen bu yöntemin uygulanabilirliği birkaç uygulama örneği ile gösterilmiştir.

Anahtar Kelimeler: Ayrık Olaylı Sistemler, Esnek Üretim Sistemleri, Denetleyici Kontrol, Sırasal Devamlılığı Koruma Modelleri.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Prof. Dr. Klaus Werner Schmidt. Also, I am very grateful to him, for his leading role in the support, continued encouragement and guidance to achieve the correct end during the course of the study and thesis.

I also thank and appreciate the Scientific and Technological Research Council of Turkey (TÜBİTAK), in the provision of material and moral support represented by full PhD scholarship. It's all thanks to Prof. Dr. Klaus Werner Schmidt and Dr. Zeki Uğurata Kocabıykoğlu for supporting me to receive this scholarship, all thanks and respect to them.

Many thanks also to my Co-Supervisor Assoc. Prof. Dr. Hüsnü Deniz Başdemir for their able guidance and support in completing my thesis and a very special thanks to Prof. Dr. Ece Güran Schmidt for providing me with all the facility that was required.

And also, I thank Çankaya University, and especially the Department of Electronic and Communication Engineering, as well as the Department of Electrical and Electronics Engineering at the Middle East Technical University, for their support and for creating the appropriate requirements of laboratories and equipment necessary during periods of study and research.

Finally, all the love and respect to my beloved mother and dear father for their love, support and encouragement during my studies.

TABLE OF CONTENTS

STATEMENT OF NON-PLAGIARISM	iii
ABSTRACT	iv
ÖZ	vi
ACKNOWLEDGMENTS	viii
TABLE OF CONTENTS	ix
LIST OF TABLES	xi
LIST OF FIGURES	xii
CHAPTERS	
1 INTRODUCTION	1
2 BASIC NOTATION	4
2.1 Discrete Event Systems	4
2.1.1 Formal Language	5
2.1.2 Automata	7
2.2 Supervisory Control	10
3 MOTIVATION AND PROBLEM STATEMENT	13
3.1 Motivating Examples	13
3.1.1 Conveyor Belt with Multiple Products	13
3.1.2 Small Production System	16
3.1.2.1 Scenario with a single product type and capacity of one product on C	17
3.1.2.2 Scenario with two product types and capacity of one product on C	18
3.1.2.3 Scenario with two product types and capacity for two products on C	19

3.1.2.4	Scenario with three product types and capacity for two products on C	19
3.2	Discussion and Research Problem	21
4	ORDER-PRESERVING LANGUAGES	23
4.1	Notation and Definitions	23
4.2	Supremal Order-preserving Language	26
4.3	Composition of Order-preserving Languages	31
4.4	Usage of Composed Order-preserving Models in Supervisory Control	43
5	MODELING OF MODULAR PRODUCTION SYSTEMS	46
5.1	Modular Production Systems and Order-preserving Models	46
5.2	Order-preserving Models	49
5.3	Illustrative Example	50
5.3.1	FMS Example	50
5.3.2	Order-preserving Model for the Example System	50
5.3.3	Supervisor Computation for the Example System	52
6	APPLICATION EXAMPLES	56
6.1	Flexible Manufacturing Systems Example	56
6.1.1	FMS Description	56
6.1.2	Modeling of the FMS	58
6.1.3	Composed Order-preserving Models for the FMS	61
6.2	Simulation of Flexible Manufacturing Systems	64
6.2.1	Small FMS Model	64
6.2.2	FMS with a Long Conveyor Belt	69
7	CONCLUSIONS	74
	REFERENCES	76
	CURRICULUM VITAE	79

LIST OF TABLES

Table 2.1	Design procedure for the supervisory control problem.	12
-----------	---	----



LIST OF FIGURES

Figure 2.1	Example finite state automaton G	7
Figure 3.1	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 three; (g) two connected CBs with different product types.	14
Figure 3.2	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 order-preserving.	15
Figure 3.3	Example system with one conveyor (C) and two machines (M_1 and M_2).	16
Figure 3.4	Automata models for the conveyor belt (C) and machines (M_1 and M_2).	17
Figure 3.5	Conveyor belt: (a) One product type and capacity one; (b) two product types and capacity one; (c) two product types and capacity two; (d) three product types and capacity two.	17
Figure 3.6	Models for the case in Fig. 3.5 (a).	18
Figure 3.7	Models for the case in Fig. 3.5 (b).	18
Figure 3.8	Models for the the case in Fig. 3.5 (c).	19

Figure 3.9	Models for the the case in Fig. 3.5 (d).	20
Figure 4.1	Example automata	25
Figure 4.2	Order-preserving model of a CB with three product types P1, P2, P3 and capacity $C = 2$.	31
Figure 4.3	Example automata	33
Figure 4.4	Composition of order-preserving languages.	34
Figure 4.5	Sufficiency of Theorem 4.	42
Figure 5.1	Schematic of the FMS.	51
Figure 5.2	Robot models.	51
Figure 5.3	Model of machine M1.	52
Figure 5.4	Model of machine M2.	52
Figure 5.5	Two examples of deadlock situations.	53
Figure 5.6	Model of the machines M1 and M2 without preserving the product order.	54
Figure 5.7	Example product path: Order-preserving case (left) and case with arbitrary product order (right).	55
Figure 6.1	FMS overview.	57
Figure 6.2	Robot models.	58
Figure 6.3	Order-preserving machine models.	59
Figure 6.4	Buffer models.	60
Figure 6.5	Machine models.	61
Figure 6.6	Workcell outline.	62
Figure 6.7	Workcell models.	63
Figure 6.8	FMS model with different production components.	64
Figure 6.9	Machine, stack feeder and exit slide models of the small FMS.	65
Figure 6.10	Rotary table model of the small FMS.	65
Figure 6.11	Order-preserving specification for the small FMS.	66
Figure 6.12	Snapshots of the FMS operation.	67

Figure 6.13	Snapshots of the FMS operation (continued).	68
Figure 6.14	FMS model with different production components.	69
Figure 6.15	Model of RT1.	69
Figure 6.16	Model of RT2.	70
Figure 6.17	CB and exit slide models.	70
Figure 6.18	Order-preserving specification for the small FMS.	71
Figure 6.19	Snapshots of the FMS operation.	72
Figure 6.20	Snapshots of the FMS operation (continued).	73





CHAPTER 1

INTRODUCTION

Discrete event system (DES) models are used for systems that reside on a *discrete state space* and whose state evolution depends on the occurrence of discrete *events* [1]. Examples for DES are human-made systems such as manufacturing systems, transportation systems or communication networks [2, 3, 4, 5, 6]. In particular, DES models are suitable for Flexible manufacturing systems (FMS). FMS have the ability of manufacturing different types of products using a given hardware setup with various production components such as machines, robots and conveyor belts [7, 8, 9, 10, 11]. In an FMS, it is generally desired to move products along pre-defined paths through the FMS and use pre-specified production components for processing these products in a given sequence [12, 13, 14]. Considering that FMS can process different types of products using the same production components, it is important to keep track of the products traveling through and FMS and to avoid blocking situations in case different products require processing by the same production component.

The logic control of FMS can be carried out in the framework of supervisory control for discrete event systems (DES) [1]. In this formal framework, the desired production sequences are realized by a supervisor that is designed based on a formal DES model of the FMS and a formal specification of the production sequence. The existing literature considers various aspects regarding the supervisory control of FMS. The work in [15, 16, 17, 18] focuses on the avoidance of deadlocks or forbidden states in FMS and [12, 13, 19, 20] develop modular and hierarchical

methods for the efficient computation of supervisors for FMS. Furthermore, there is recent work such as [14] that proposes the concept of distinguishers in order to facilitate the modeling of FMS with different product types. In this context, it has to be noted that, although an FMS should keep track of the sequence of products traveling through the FMS, the existing methods avoid this necessity by certain modeling tricks that facilitate the FMS modeling and that restrict generality.

Accordingly, the main subject of this thesis is the modeling of an additional property of FMS that has not been addressed in previous work on the supervisory control of FMS. In particular, it is possible that production components can hold multiple products and then process them sequentially. In this case, it is required that a product that enters the production component first will also be processed first and then leave the production component first. That is, if a production component is able to process different product types, it is necessary to keep track of the different product types entering and leaving such production component. Nevertheless, this case does not appear in the existing literature. For FMS with different product types it is either the case that these product types have independent paths in the FMS [9, 14, 21], the product paths are defined such that it is not necessary to remember the product order [7, 12, 13, 19, 20, 22, 23] or the product order is not taken into account and it is implicitly assumed that the production component knows which product is currently transported [8, 15, 16].

In order to address this issue, the thesis first determines scenarios under which a suitable model for a production component needs to keep track of the product type and the order of products entering and leaving. Then, a formal modeling framework denoted as "order-preserving languages" is introduced and important properties of order-preserving languages are studied. In particular, it is shown that order-preserving languages are closed under arbitrary union and the composition of order-preserving languages again yields an order-preserving language.

As an extension of the theoretical investigation of order-preserving languages, the thesis further develops a general method for algorithmically constructing the required order-preserving models of production components. This method takes into account the neighborhood relationship between production components and

is applicable to production components that can hold an arbitrary number of products and that can process an arbitrary number of product types. We demonstrate the practicability of the proposed modeling method by different FMS examples with multiple products and overlapping product paths.

The remainder of the thesis is organized as follows. Chapter 2 introduces the basic notation regarding DES and supervisory control. Chapter 3 formulates the problem addressed in this thesis based on a variety of motivating examples. Then, Chapter 4 introduces the new modeling formalism of order-preserving languages and derives their most important properties. The proposed order-preserving modeling method using finite state automata is developed in Chapter 5 and applied to various FMS examples in Chapter 6. Chapter 7 gives conclusions and states ideas for future work.

CHAPTER 2

BASIC NOTATION

This chapter gives the necessary background information about modeling and supervisory control of discrete event systems (DES) that are studied in this thesis. First, Section 2.1 defines DES and introduces the modeling formalisms of formal languages and finite state automata. Then, Section 2.2 discusses the supervisory control of DES.

2.1 Discrete Event Systems

The expression "discrete event system" (DES) was introduced in the early 1980s to model an increasingly important class of human-made dynamic systems with a discrete state space and state transitions that are given by the asynchronous occurrence of discrete events. Such systems are encountered in a variety of fields and have been successfully employed in many areas, for example in manufacturing systems and communication networks [1, 24]. The operation of a DES is in principle governed by sequences of events, whereby the order of events is most relevant. The exact timing of events is nevertheless not important. A DES model can hence very well describe systems with activities such as turning some device "On ,Off", sending one or multiple message packets, or detecting if an object arrives at a certain location.

DES can be characterized by the following distinctive properties:

- A DES has a finite or countably infinite number of discrete states
- Each change of state occurs at a discrete time instant. State changes are called *transitions*
- Each transition is driven by the occurrence of an event
- A DES spends time only in states. That is, state transitions occur instantaneously (they do not need time).

A light switch is a simple example of a DES. The switch has two discrete states, which can be identified as "On" and "Off". There are further two possible events `switchOn` and `switchOff`. If the light switch is for example in state "Off", the event `switchOn` can occur, and the light switch performs an instantaneous transition to the state "On". Analogously, in state "On", a transition with event `switchOff` leads to state "Off".

In the established literature, the behavior of a DES is modeled by a formal language [1]. We next introduce the concept of a formal language.

2.1.1 Formal Language

The behavior of discrete event systems is represented by formal languages over finite set of events, also called *alphabet* Σ . This set consists of all the events that can possibly happen in a given DES.

A *string* (or trace) is a finite sequence of events from Σ . The length of a string s , denoted by $|s|$ is given by the number of events in s . The empty string, denoted by ϵ , is the string with zero length (i.e., $|s| = 0$). Each element $\sigma \in \Sigma$ is denoted as an event, Σ^* denotes the set of all finite strings over Σ . whereby the \star -operation is called *Kleene Closure*.

Considering the light switch example, the alphabet is given by $\Sigma = \{\text{switchOn}, \text{switchOff}\}$. A possible sequence of events is $s = \text{switchOn } \text{switchOff } \text{switchOn}$ with length $|s| = 3$. The Kleene closure of Σ in this case is $\Sigma^* = \{\epsilon, \text{switchOn}, \text{switchOff}, \text{switchOnswitchOn}, \text{switchOnswitchOff}, \text{switchOffswitchOn},$

`switchOff switchOff, ..., }`. Note that not all strings in Σ^* must be possible in the DES.

The concatenation of two strings $s_1 s_2 \in \Sigma^*$ is defined as $s_1, s_2 \in \Sigma^*$. $s_1 \leq s$ indicates that s_1 is a *prefix* of s and $\epsilon \in \Sigma^*$ is the empty string. With a slight abuse of notation, we write $\sigma \in s$ for a string $s \in \Sigma^*$ and an event $\sigma \in \Sigma$ if σ appears in s .

For any string $s = \sigma_1, \dots, \sigma_{|s|} \in \Sigma^*$ with $\sigma_i \in \Sigma$ for $i = 1, \dots, |s|$, we further write $\text{pre}_k(s) = \sigma_1 \dots \sigma_k$ for the prefix of s with the first k events and $\text{suf}_k(s) = \sigma_{|s|-k+1} \dots \sigma_{|s|}$ for the suffix of s with the last k events. In particular, $s = \text{pre}_k(s) \text{suf}_{|s|-k}(s)$ for any $k \leq |s|$.

A formal *language* over Σ is a subset $L \subseteq \Sigma^*$. Then,

$$\bar{L} := \{s_1 \in \Sigma^* \mid \exists s \in L \text{ s.t. } s_1 \leq s\}$$

defines the *prefix closure* of L , and L is called *prefix closed* if $L = \bar{L}$.

Consider two alphabets $\hat{\Sigma}, \Sigma$ such that $\hat{\Sigma} \subseteq \Sigma$. The *natural projection* erases all events in $\Sigma \setminus \hat{\Sigma}$ (\setminus is the set difference) from strings $s \in \Sigma^*$. This function is defined as $p : \Sigma^* \rightarrow \hat{\Sigma}^*$ iteratively such that

$$\begin{aligned} p(\epsilon) &= \epsilon \\ p(\sigma) &= \begin{cases} \sigma & \text{if } \sigma \in \hat{\Sigma} \\ \epsilon & \text{otherwise} \end{cases} \\ p(s\sigma) &= p(s)p(\sigma) \end{aligned}$$

For example, if we are only interested in the event `switchOn` of the light switch, we can project each string to the alphabet $\hat{\Sigma} = \{\text{switchOn}\}$ and make occurrences of `switchOff` invisible. For the string $s = \text{switchOn switchOff switchOn}$, the projection gives $p(s) = \text{switchOn switchOn}$.

The set-valued inverse of p is written as

$$p^{-1} : \hat{\Sigma} \rightarrow 2^{\Sigma^*}, p_i^{-1}(t) := \{s \in \Sigma^* \mid p_i(s) = t\}$$

Consider two languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ for alphabets Σ_1, Σ_2 . Using the

natural projection, the *synchronous composition* $L_1 || L_2 \subseteq (\Sigma_1 \cup \Sigma_2)^*$ of L_1 and L_2 is computed as $L_1 || L_2 = p_1^{-1}(L_1) \cap p_2^{-1}(L_2)$.

2.1.2 Automata

A very widely used compact model for languages is the automaton. It allows to describe and study the structure of DES and the automaton graph visualizes the behavior of DES. Usually, a DES is modeled by a finite state automaton.

A finite state automaton is a 5-tuple $G = (X, \Sigma, \delta, x_0, X_m)$ with

- the finite set of *states* X ;
- the finite alphabet of *events* Σ ;
- the partial *transition function* $\delta : X \times \Sigma \rightarrow X$;
- the *initial state* $x_0 \in X$; and
- the set of *marked states* $X_m \subseteq X$.

Hereby, $\delta(x, \sigma)!$ is written if δ is defined at (x, σ) and we extend the transition function δ to a partial function on $X \times \Sigma^*$ in the usual way. That is, if $\delta(x, s)$ exists for some $s \in \Sigma^*$, $\delta(x, s)$ represents the state reached starting from state x and executing the string s .

We use the example automaton $G = (X, \Sigma, \delta, x_0, X_m)$ in Fig. 2.1 to explain the notation introduced before.

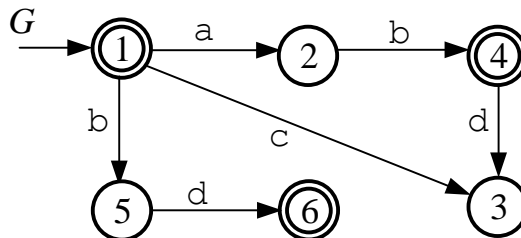


Figure 2.1: Example finite state automaton G .

G has 6 states with $X = \{1, 2, 3, 4, 5, 6\}$ and an alphabet with 4 events $\Sigma = \{a, b, c, d\}$. The transition relation δ is defined with $\delta(1, a) = 2$, $\delta(1, b) = 5$, $\delta(1, c) = 3$, $\delta(2, b) = 4$, $\delta(4, d) = 3$, $\delta(5, d) = 6$. The initial state is $x_0 = 1$ and the set of marked states is $X_m = \{1, 4, 6\}$.

The behavior of an automaton G is given by its *closed language*:

$$L(G) := \{s \in \Sigma^* \mid \delta(x_0, s)!\}$$

and its *marked language*:

$$L_m(G) := \{s \in L(G) \mid \delta(x_0, s) \in X_m\}$$

The closed language $L(G)$ contains all event sequences that follow the transitions of G starting from the initial state. The language $L_m(G)$ contains all strings of events, starting from the initial state of G and leading to a marked state in X_m . For the example automaton in Fig. 2.1, we determine the closed language $L(G) = \{\epsilon, a, ab, abd, c, b, bd\}$ and the marked language $L_m(G) = \{\epsilon, ab, bd\}$.

A finite state automaton G is said to be *nonblocking* if

$$\overline{L_m(G)} = L(G)$$

This property holds, when every string generated by G can be extended to a marked (desired) state in G . Finite state automata can be *cyclic* or *acyclic*. A cycle in a finite automaton is a sequence of states x_1, x_2, \dots, x_k (k is a natural number) such that $x_1 = x_k$ and for all $i = 1, \dots, k - 1$, there exists an event $\sigma_i \in \Sigma$ such that $\delta(x_i, \sigma_i) = x_{i+1}$. This means, it is possible to start at a state x_1 of G , follow the transitions in G and return back to x_1 . Then, an automaton G without cycles is called acyclic.

Now we introduce several relevant properties and operations for automata:

- *Accessible*: The automaton $G = (X, \Sigma, \delta, x_0, X_m)$ is *accessible*, if all states in X can be reached from the initial state x_0 . Formally, we write

$$\forall x \in X, \exists s \in \Sigma^* \text{ such that } \delta(x_0, s) = x$$

In addition, we write $Acc(G)$ for the automaton, where all states from G that are not reachable from x_0 are removed. Then, $Acc(G)$ is accessible.

- *Coaccessible*: The automaton $G = (X, \Sigma, \delta, x_0, X_m)$ is *coaccessible*, if it is possible to reach a marked state from any state in X . Formally, we write

$$\forall x \in X, \exists s \in \Sigma^* \text{ such that } \delta(x, s) \in X_m$$

It holds that a coaccessible automaton is nonblocking: $\overline{L_m(G)} = L(G)$. In addition, we write $CoAcc(G)$ for the automaton, where all states from G that are not coaccessible are removed. Then, $CoACC(G)$ is coaccessible.

- *Trim*

The automaton $G = (X, \Sigma, \delta, x_0, X_m)$ is *trim*, if it is accessible and coaccessible at the same time. We write

$$Trim(G) = Acc(CoAcc(G)) = CoAcc(Acc(G))$$

The example automaton G in Fig. 2.1 is accessible because all states in X are reachable from the initial state 1. Hence, $Acc(G) = G$. However, G is not coaccessible because there is no path from the state 3 to a marked state. Hence $CoAcc(G)$ is obtained by removing state 3 from G . Then, $CoAcc(G)$ is a strict subautomaton of G . It can also be seen that G is acyclic.

If a DES is modeled by more than one finite state automaton (for example, if the system has several components with their own automata model), the *synchronous composition* operation can be used to obtain a single automaton model of the DES. Assume two automata $G_1 = (X_1, \Sigma_1, \delta_1, x_{0,1}, X_{m,1})$ and $G_2 = (X_2, \Sigma_2, \delta_2, x_{0,2}, X_{m,2})$ are given.

The synchronous composition is written as:

$$G_{12} = (X_{12}, \Sigma_{12}, \delta_{12}, x_{0,12}, X_{m,12}) = G_1 || G_2,$$

and is defined such that

- $X_{12} = X_1 \times X_2$ (canonical product of states from X_1 and X_2)
- $\Sigma_{12} = \Sigma_1 \cup \Sigma_2$ (union of events in Σ_1 and Σ_2)
- $x_{0,12} = (x_{0,1}, x_{0,2})$

- $X_{m,12} = X_{m,1} \times X_{m,2}$
- the transition relation takes care that events in $\Sigma_1 \cap \Sigma_2$ that are shared by G_1 and G_2 are synchronized. For $(x_1, x_2) \in X_{12}$ and $\sigma \in \Sigma_{12}$:

$$\delta_{12}((x_1, x_2), \sigma) = \begin{cases} (\delta_1(x_1, \sigma), \delta_2(x_2, \sigma)) & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2 \wedge \delta_1(x_1, \sigma)! \wedge \delta_2(x_2, \sigma)! \\ (\delta_1(x_1, \sigma), x_2) & \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \wedge \delta_1(x_1, \sigma)! \\ (x_1, \delta_2(x_2, \sigma)) & \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \wedge \delta_2(x_2, \sigma)! \end{cases}$$

That is, the important point of the synchronous composition [1] is that it synchronizes the shared events (events that appear in both automata), whereas all other events can occur independent of each other. In relation to the synchronous composition of languages, it holds that $L(G_1 || G_2) = L(G_1) || L(G_2)$ and $L_m(G_1 || G_2) = L_m(G_1) || L_m(G_2)$.

Until now, we described how a DES can be modeled by formal languages and finite state automata, respectively. The supervisory control theory for DES is introduced in the next section.

2.2 Supervisory Control

The supervisory control theory for DES was first established by Ramadge and Wonham [25]. It offers a formal framework to design and implement control for DES. The control of the system is executed by allowing or preventing specific events from occurring in the plant. Such control is performed by a controller (or supervisor) while taking into consideration the necessity to ensure the desired behavior of the system.

In supervisory control, we write $\Sigma = \Sigma_c \cup \Sigma_u$ for *controllable* (Σ_c) and *uncontrollable* (Σ_u) events. We say that an automaton $S = (Q, \Sigma, \nu, q_0, Q_m)$ 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 $\sigma \in \Sigma_u$ with $s\sigma \in L(G)$ that also $s\sigma \in L(S)$. Then, $L(G) || L(S)$ and $L_m(G) || L_m(S)$ represent the closed and marked behavior of the closed-loop system $G || S$, respectively. A supervisor S is called nonblocking if the automaton $G || S$ is nonblocking.

The design of the supervisor S is based on the desired behavior of the plant. This desired behavior is usually given in the form of another automaton $C = (Y, \Sigma, \beta, y_0, Y_m)$ and $K = L_m(C)$ is called the *specification* language for the control problem. The meaning of K is, that it contains all strings that are desirable. On the other hand, a supervisor should disable all strings in $L(G)$ that do not belong to K . The question if this task is possible is answered by the *controllability* condition.

A language $K \subseteq L_m(G)$ is said to be controllable for $L(G)$ and Σ_u if

$$\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}$$

In this expression, \overline{K} is the prefix-closure of K and $\overline{K}\Sigma_u$ represents the set of all strings that start with a prefix in \overline{K} concatenated with an uncontrollable event in Σ_u . In words, K is *controllable* with respect to G and Σ_u if, whenever a string $s \in \overline{K}$ that is allowed by the specification can be extended by an uncontrollable event in the plant ($s\sigma \in L(G)$) the extended string must again belong to the specification ($s\sigma \in \overline{K}$). This is necessary, because σ could not be disabled by a supervisor after s . There exists a supervisor S such that $L_m(G||S) = K$ if and only if K is controllable for $L(G)$ and Σ_u [25].

In case, K is not controllable for $L(G)$ and Σ_u , the supervisor will implement the *supremal controllable sublanguage* of K . We write

$$L_m(S||G) = \text{SupC}(K, G, \Sigma_u)$$

That is, $\text{SupC}(K, G, \Sigma_u)$ includes all sublanguages of K that are controllable with respect to G and Σ_u . It is ensured that such supervisor is nonblocking and maximally permissive if $\text{SupC}(K, G, \Sigma_u) \neq \emptyset$ [24].

In summary, the supervisory control theory allows to algorithmically compute a supervisor S for a DES plant G such that a given language specification is fulfilled, which means that $L_m(G||S) \subseteq K$. The design problem is summarized in the following table.

Table 2.1: Design procedure for the supervisory control problem.

Given	Desired	Solution
Plant G Specification K Uncontrollable events Σ_u	Find supervisor S such that $L_m(G S) \subseteq K$	Compute $\text{SupC}(K, G, \Sigma_u)$ Use supervisor S with $L_m(G S) = \text{SupC}(K, G, \Sigma_u)$



CHAPTER 3

MOTIVATION AND PROBLEM STATEMENT

In this chapter, we introduce the main issues addressed in this thesis. First, Section 3.1 gives several motivating examples. Then, Section 3.2 discusses the main observations and states the main problem considered in this thesis.

3.1 Motivating Examples

The main focus of this thesis is the generation of DES models that retain the order of products for production systems. In this section, we illustrate the need for such models by two example scenarios. The first scenario in Section 3.1.1 considers the modeling of conveyor belts with multiple products. As an extension, Section 3.1.2 discusses DES models for a small production systems with one conveyor belt and two machines.

3.1.1 Conveyor Belt with Multiple Products

The work in this thesis is motivated by an observation from modeling complex flexible manufacturing systems (FMS) where (i) different product types can be processed on the same production component and (ii) production components might have different capacities. That is, such production component is 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. 3.1.

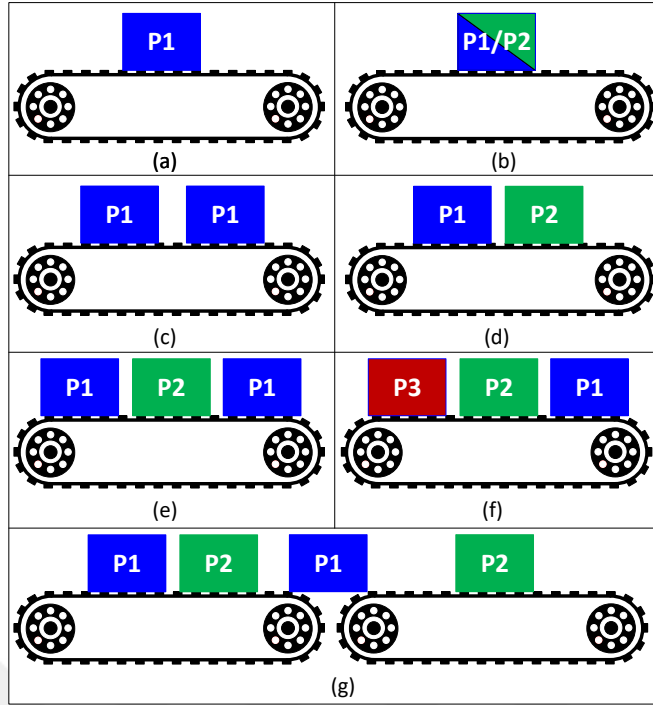


Figure 3.1: CBs with different product types and capacities: (a) one product type and capacity one; (b) two product types and capacity one; (c) one product type and capacity two; (d) two product types and capacity two; (e) two product types and capacity three; (f) three product types and capacity three; (g) two connected CBs with different product types.

We next discuss the cases in the different parts of Fig. 3.1 together with possible models. Hereby, we introduce events such as in_{P_1} , in_{P_2} , \dots for products entering the CB and out_{P_1} , out_{P_2} , \dots for products leaving the CB. Part (a) and (b) show the case of a CB with one and two product types and a capacity of one product. That is, the CB in part (a) will always take a single product, whose type is unique. Differently, the CB in part (b) will take a single product but its type can be either P1 or P2. In addition, part (c) considers the case of one product type and a capacity of two. This means that two products can be present on the CB simultaneously. However, since these products have the same type P1, it is clear that always a product of type P1 will leave the CB. The cases (a), (b) and (c) can be straightforwardly modeled by the automata in Fig. 3.2 (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 (case (a) and (c)) or is preserved by these models (case (b)).

Differently, the modeling becomes more involved in part (d) of Fig. 3.1. 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. For example, if two products with type P1 and P2 enter the CB in the sequence P1-P2 (represented by the string $in_{P1}in_{P2}$), they also must leave the CB in this sequence (represented by the string $out_{P1}out_{P2}$). In this case, the model in Fig. 3.2 (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 $in_{P1}in_{P2}out_{P1}$). A model that captures the correct order of product types entering and leaving the CB is shown in Fig. 3.2 (d-2). Here, only out_{P1} is possible in state 5 (after $in_{P1}in_{P2}$) and only out_{P2} is possible in state 6 (after $in_{P2}in_{P1}$). More complicated scenarios can easily be envisaged by increasing the capacity (as in Fig. 3.1 (e)) and/or the number of product types (as in Fig. 3.1 (f)). Moreover, it is possible to consider the connection of multiple CBs that keep the order of product types as in Fig. 3.1 (g).

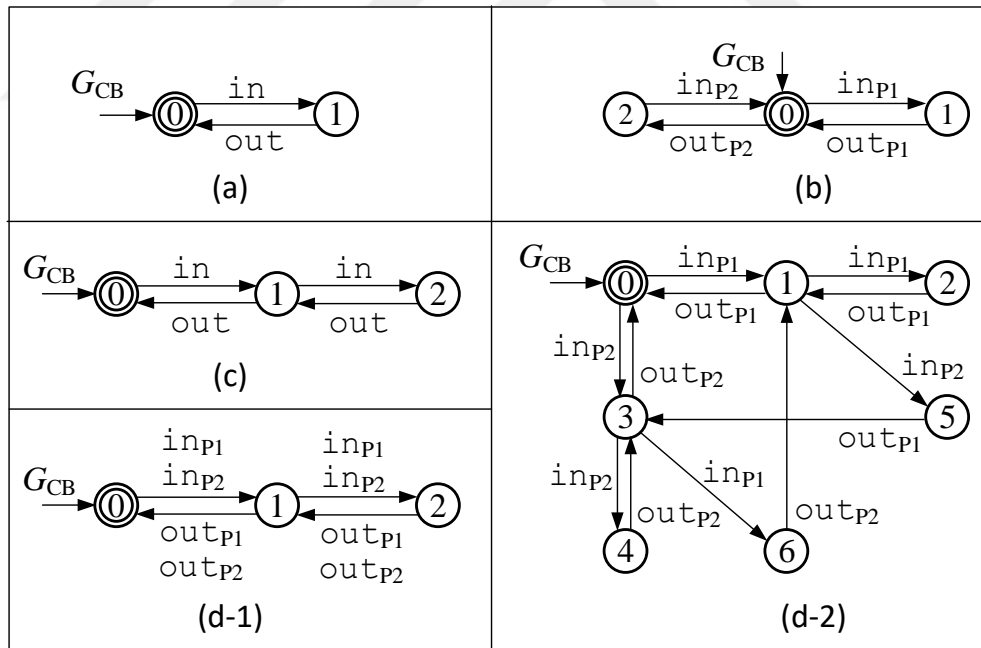


Figure 3.2: CB models for different cases: (a) one product type and capacity one; (b) two product types and capacity one; (c) one product type and capacity two; (d-1) two product types and capacity two with arbitrary order; (d-2) two product types, capacity two and order-preserving.

3.1.2 Small Production System

In the second motivating example, we take into account that production systems are generally composed of multiple production components. That is, we study the simple production system with one conveyor belt (C) and two machines (M1 and M2) in Fig. 3.3.

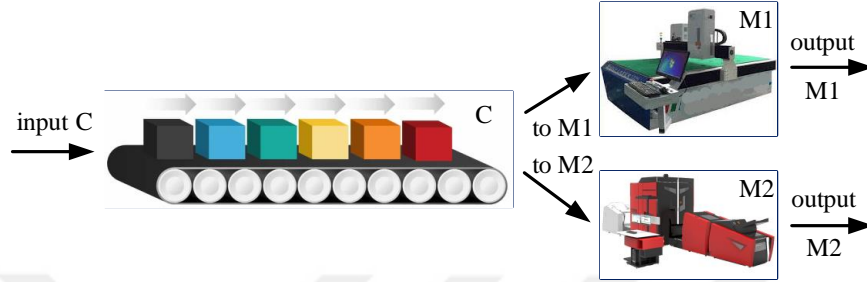


Figure 3.3: Example system with one conveyor (C) and two machines (M_1 and M_2).

In analogy to the previous section, we focus on the case where the conveyor belt (C) is potentially long and can hold multiple products simultaneously. Products are fed to C from the left and are then transported to one of the machines M1 or M2 on the right hand side of the figure. Each product is then processed in the respective machine and leaves the example system on the right hand side.

In this simple setup, the common automaton model G_C of the conveyor and the automata models G_{M1} and G_{M2} of the machines are as given in Fig. 3.4. That is, the conveyor receives products with the event inC and delivers products towards the machines with the event outC . If C can hold at most n products simultaneously (that is, the capacity of C is n), G_C has $n+1$ states to keep memory of the number of products in C. The machines M1 and M2 obtain products with the event inM1 and inM2 , respectively. After processing products are delivered to the outside with outM1 and outM2 .

Although such models are frequently used in the modeling of manufacturing systems, flexible manufacturing systems (FMS) [26] as well as reconfigurable manufacturing systems (RMS) [27], we again argue that these models are not suitable if there are different product types (with different processing requirements).

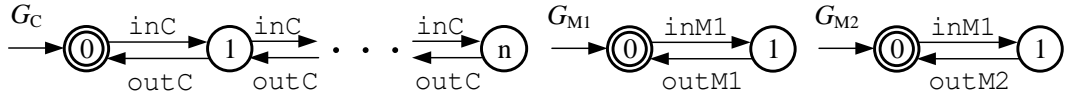


Figure 3.4: Automata models for the conveyor belt (C) and machines (M1 and M2).

To this end, we next discuss four relevant scenarios and their effect on the conveyor model as depicted in Fig. 3.5. Different from the example in the previous section, we now consider that the conveyor belt has neighboring production components (M1 and M2) with different processing capabilities. That is, it is important to move the correct product type to the correct machine.

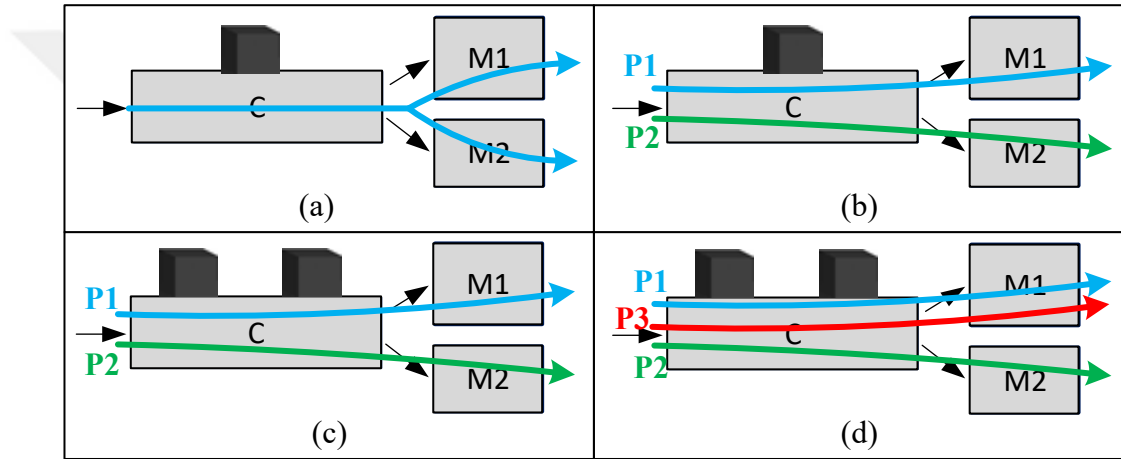


Figure 3.5: Conveyor belt: (a) One product type and capacity one; (b) two product types and capacity one; (c) two product types and capacity two; (d) three product types and capacity two.

3.1.2.1 Scenario with a single product type and capacity of one product on C

Fig. 3.5 (a) shows the case where there is a single product type and C has capacity for a single product. It is further desired that any product is delivered to one of the machines arbitrarily. In this case, it needs to be taken into account that (i) there is no difference between products that enter C; (ii) any product that enters C can leave to two different machines. Accordingly, input of a product to C can be modeled by the event inC , whereas product output has to be modeled by two

different events $C-M1$ (from C to $M1$) and $C-M2$ (from C to $M2$) as is shown in Fig. 3.6. Similarly, the model of the machines in Fig. 3.4 has to be adjusted such that $inM1$ is replaced by $C-M1$ and $inM2$ is replaced by $C-M2$ in order to match the events defined for C .

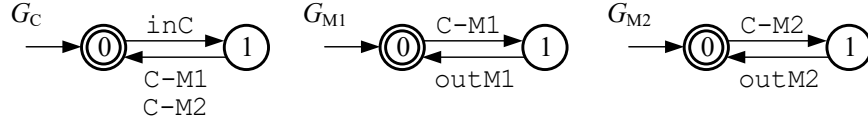


Figure 3.6: Models for the case in Fig. 3.5 (a).

3.1.2.2 Scenario with two product types and capacity of one product on C

We next investigate the scenario in Fig. 3.5 (b). Here, the conveyor can hold a single product but there are two product types. Product $P1$ (blue) needs to be processed by $M1$, whereas product $P2$ (green) has to be processed by $M2$. In this case, it needs to be taken into account that (i) different products enter C ; (ii) depending on the product type in C , the product leaves to $M1$ or $M2$. That is, the model of C in Fig. 3.6 is not suitable since it cannot distinguish products entering C . In order to distinguish the product type, a refinement of the event alphabet of G_C is required by introducing separate events for the different product types. That is, instead of $\Sigma_C = \{inC, C-M1, C-M2\}$, we use $\Sigma_C = \{inC_{P1}, inC_{P2}, C-M1, C-M2\}$. In addition, the model needs to respect the order of products entering and leaving C . In particular, it is not possible that $P2$ leaves C ($C-M2$) after $P1$ enters C (inC_{P1}) and vice versa. A suitable model for this scenario is shown in Fig. 3.7. Here, G_C expresses that each event $C-Mi$ is only possible after the respective event inC_{Pi} , $i \in \{1, 2\}$.

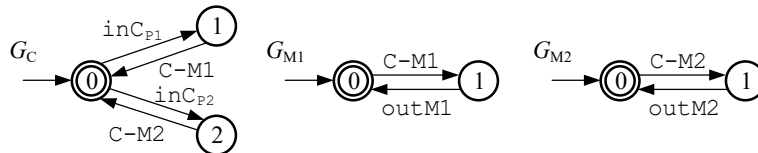


Figure 3.7: Models for the case in Fig. 3.5 (b).

3.1.2.3 Scenario with two product types and capacity for two products on C

A more complicated modeling problem is encountered in the scenario in Fig. 3.5 (c). Here, two different product types P1 and P2 are produced and C can hold up to two products. In this case, it needs to be taken into account that (i) different products enter C; (ii) depending on the product type in C, the product leaves to M1 or M2; (iii) the order of products entering and leaving C must be preserved in the model. Similar to the previous case, it is again necessary to refine the event alphabet of G_C as $\Sigma_C = \{\text{inC}_{P1}, \text{inC}_{P2}, \text{C-M1}, \text{C-M2}\}$. In addition, the model has to remember the order in which products enter the system since products have to leave C in the same order. A suitable model for this purpose is shown in Fig. 3.8. Here, G_C has one state for each possible product combination on C as is indicated by the color code in the respective states (P1 – blue; P2 – green). For example, it can be seen that the input order P1, P2 of products (string $\text{inC}_{P1} \text{inC}_{P2}$) leads to a state where only P1 can exit C to M1 (event C-M1), whereas P2 has to wait until P1 leaves C.

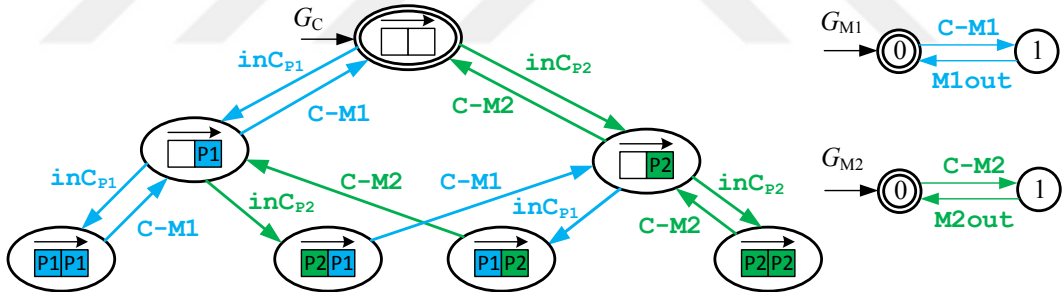


Figure 3.8: Models for the the case in Fig. 3.5 (c).

3.1.2.4 Scenario with three product types and capacity for two products on C

The next example shows that the modeling problem need not be restricted to a single component of a production system. To this end, we consider the scenario in Fig. 3.5 (d). Here, there are three product types P1, P2, P3 and C has a

capacity of two products. P2 is delivered to M2, whereas both P1 and P3 are delivered to M1. That is, P1 and P3 share the same path within the example system but might follow a different path after leaving the example system. In this case, it needs to be taken into account that (i) different products enter C; (ii) depending on the product type in C, the product leaves to M1 or M2; (iii) the order of products entering and leaving C and M1 must be preserved in the model. Since there are three products in C and two products in M1, we refine the respective alphabets as $\Sigma_C = \{\text{inC}_{P_1}, \text{inC}_{P_2}, \text{inC}_{P_3}, \text{C-M1}_{P_1}, \text{C-M1}_{P_3}, \text{C-M2}_{P_2}\}$, $\Sigma_{M1} = \{\text{C-M1}_{P_1}, \text{C-M1}_{P_3}, \text{outM1}_{P_1}, \text{outM1}_{P_3}\}$ and $\Sigma_{M2} = \{\text{C-M2}_{P_2}, \text{outM2}_{P_2}\}$. In addition, the model has to remember the order in which three different products enter C (capacity 2) and the order in which two different products enter M1 (capacity 1). A suitable model for this purpose is shown in Fig. 3.9. Similar to Fig. 3.8, G_C has one state for each possible product combination on C. Since there are more product types, more combinations have to be considered. Finally, the model for M1 has the same structure as the model for C in Fig. 3.7 since there are two products types and the capacity is one.

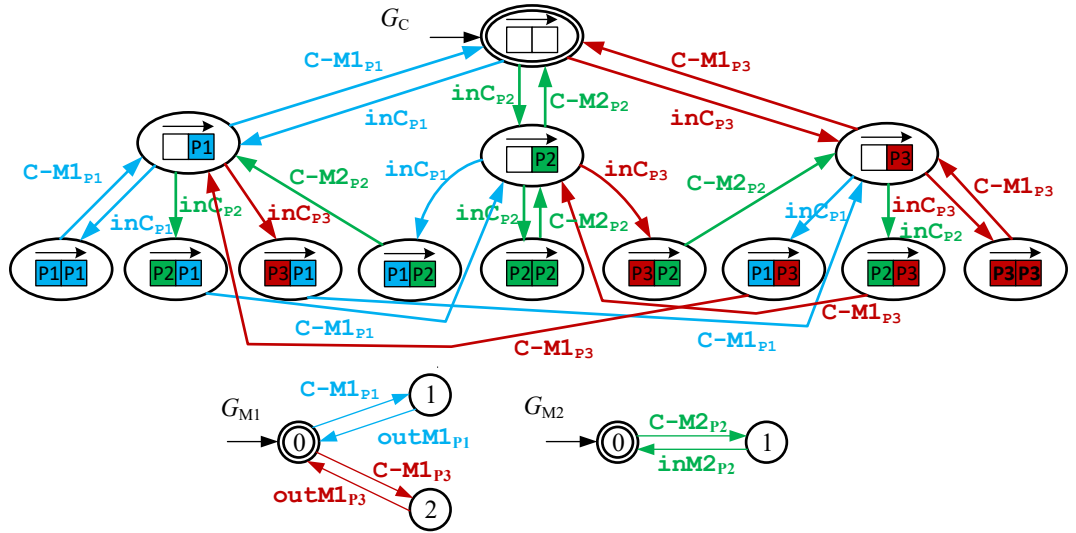


Figure 3.9: Models for the the case in Fig. 3.5 (d).

3.2 Discussion and Research Problem

The main purpose of the example scenarios in Fig. 3.1 and Fig. 3.5 is to clarify the effect of different product types and different capacities of production components on the structure of the respective automaton model. In particular, it can be observed from the scenario in Fig. 3.1 (b) and Fig. 3.5 (b) that DES models need to distinguish products if there is more than one product type. In addition, the scenarios in Fig. 3.1 (d) to (f) and Fig. 3.5 (c) indicate that the DES model needs to remember the order of products if there are multiple product types and the capacity of the production component is greater than one. Finally, the scenarios in Fig. 3.1 (g) and Fig. 3.5 (d) show that the information about capacity and different product types has to be incorporated in the DES models of all relevant production components.

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

It is interesting to note that, although the existing literature considers the control of production systems with different product types [7, 8, 9, 12, 13, 14, 15, 16, 19, 20, 21, 22, 23], none of the existing works develops DES models that preserve the order of products. In order to avoid the need for order-preserving 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 production systems, where different product types use disjoint paths through the system and are hence processed by different production components [9, 14, 21]. In this case, products need not be distinguished and models as in Fig. 3.2 (a) and (c) are suitable. In the second case, it is possible that the paths of different products intersect [7, 8, 12, 13, 15, 16, 19, 20,

22, 23]. Although this implies that the same production component is passed by multiple products, these research works again make simplifying assumptions. On the one hand, [7, 12, 13, 19, 20, 22, 23] choose the control specification such that remembering the product order is straightforward. Specifically, this is achieved by using production components, whose capacity does not exceed one. On the other hand, [8, 15, 16] allow for production components with multiple product types and a capacity that is greater than one. However, these papers use models as in Fig. 3.2 (d-1) for such production components. That is, the order of products entering and leaving the production component is not taken into account. This means that knowledge about which product leaves such production component must be implicitly known. In a real application this implies that sensors need to be installed to detect the product type leaving such production component. In addition, using a model as in Fig. 3.2 (d-1) for an order-preserving production component means that the model contains unnecessary behavior.

Accordingly, this thesis focuses on the problem of developing a formal framework for order-preserving DES models of production components with multiple product types and capacities that are greater than one. Parts of the work are published in [28] and are under preparation in [29].

CHAPTER 4

ORDER-PRESERVING LANGUAGES

The previous chapter motivates the need for order-preserving models in DES. This chapter defines and analyzes order-preserving DES models in the framework of formal languages. First, Section 4.1 introduces the required notation for the definition of order-preserving languages and then determines their basic properties. Then, Section 4.2 shows the existence of a supremal order-preserving language and Section 4.3 studies the composition of order-preserving languages. Finally, Section 4.4 shows that reduced order-preserving models can be obtained after composition.

4.1 Notation and Definitions

We consider an alphabet Σ that is divided into disjoint sets of input events Σ^{in} and output 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 $p^{\text{in}} : \Sigma^* \rightarrow (\Sigma^{\text{in}})^*$ and $p^{\text{out}} : \Sigma^* \rightarrow (\Sigma^{\text{out}})^*$. In addition, we define a bijective input/output mapping $m : \Sigma^{\text{in}} \rightarrow \Sigma^{\text{out}}$ such that for each input event $\alpha \in \Sigma^{\text{in}}$, $m(\alpha) \in \Sigma^{\text{out}}$ denotes the corresponding output event. We further write $m^{-1} : \Sigma^{\text{out}} \rightarrow \Sigma^{\text{in}}$ for the inverse mapping. Relating this definition to the discussion about production systems in the previous section, an input event $\alpha \in \Sigma^{\text{in}}$ and output event $m(\alpha)$ represent a product entering and leaving a production component, respectively.

Given the above notation, we introduce the new notion of an order-preserving

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 $s \in \Sigma^*$ is denoted as *order-preserving* if

$$m(p^{\text{in}}(s)) = p^{\text{out}}(s). \quad (4.1)$$

A language $L \subseteq \Sigma^*$ is denoted as *order-preserving* if it holds for all $s \in L$ that s is order-preserving.

That is, each order-preserving string has the same order of input events and corresponding output events. In particular, it must be the case that each order-preserving string $s \in \Sigma^*$ 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 order-preserving string s must fulfill

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

This means that output events can only occur after the corresponding input event.

For illustration, we consider the languages $L_1 = L_m(G_1)$ and $L_2 = L_m(G_2)$ of the automata G_1 and G_2 in Fig. 4.1. Here, we assume that $\Sigma = \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{b}_1, \mathbf{b}_2\}$, $\Sigma^{\text{in}} = \{\mathbf{a}_1, \mathbf{a}_2\}$, $\Sigma^{\text{out}} = \{\mathbf{b}_1, \mathbf{b}_2\}$ and $m(\mathbf{a}_1) = \mathbf{b}_1$, $m(\mathbf{a}_2) = \mathbf{b}_2$. For example, it holds that $s_1 = \mathbf{a}_1\mathbf{a}_2\mathbf{b}_1\mathbf{b}_2 \in L_1$ and $s_2 = \mathbf{a}_2\mathbf{b}_2\mathbf{a}_1\mathbf{a}_2\mathbf{b}_1\mathbf{b}_2 \in L_2$ are order-preserving strings. Specifically, $m(p^{\text{in}}(s_1)) = m(\mathbf{a}_1\mathbf{a}_2) = \mathbf{b}_1\mathbf{b}_2 = p^{\text{out}}(s_1)$ and $m(p^{\text{in}}(s_2)) = m(\mathbf{a}_2\mathbf{a}_1\mathbf{a}_2) = \mathbf{b}_2\mathbf{b}_1\mathbf{b}_2 = p^{\text{out}}(s_2)$. It can be further verified that L_2 is an order-preserving language. Nevertheless, it is the case that L_1 is not order-preserving. This can be seen by looking at the string $s_3 = \mathbf{a}_1\mathbf{a}_2\mathbf{b}_2\mathbf{b}_1 \in L_1$ with $m(p^{\text{in}}(s_3)) = \mathbf{b}_1\mathbf{b}_2 \neq p^{\text{out}}(s_3) = \mathbf{b}_2\mathbf{b}_1$.

Referring to Definition 1, we write

$$\mathcal{L}^{\text{OP}}(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m) = \{L \subseteq (\Sigma^{\text{in}} \cup \Sigma^{\text{out}})^* \mid L \text{ is order-preserving}\} \quad (4.2)$$

for the set of all order-preserving languages. In addition, we introduce the notion of the *capacity* of an order-preserving language.

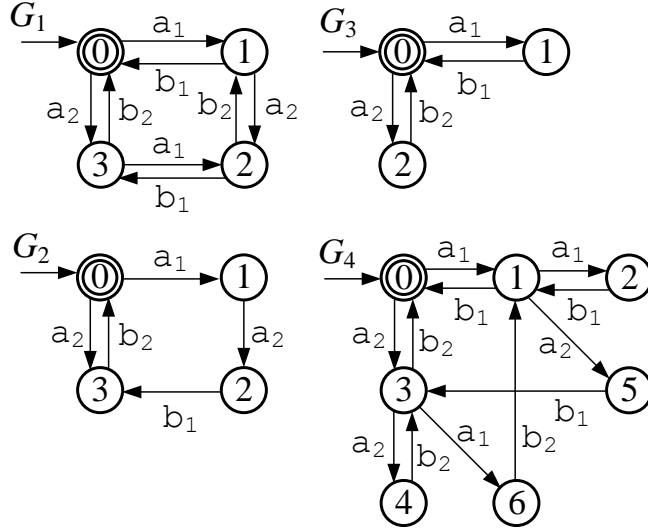


Figure 4.1: Example automata

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

$$c(s) = \max_{s' \leq s} \{|p^{in}(s')| - |p^{out}(s')|\}. \quad (4.3)$$

Extending this notion to languages, the capacity of any order-preserving language $L \in \mathcal{L}^{OP}(\Sigma^{in}, \Sigma^{out})$ is defined as

$$c(L) = \max_{s \in L} \{c(s)\}. \quad (4.4)$$

That is, the capacity of an order-preserving string $s \in \Sigma^*$ 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 order-preserving language is given by the maximum capacity of any string $s \in L$.

Consider the languages $L_2 = L_m(G_2)$ and $L_3 = L_m(G_3)$ of the automata G_2 and G_3 in Fig. 4.1. 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 $a_1 a_2$. Differently, $c(L_3) = 1$ since each input event a_i , $i = 1, 2$, is directly followed by the corresponding output event b_i .

In order to relate the notion of capacity to practical systems, we consider pro-

duction systems as discussed in the previous section. Here, input events could be identified with products of certain types entering the system, whereas output events can characterize the corresponding products exiting the system. Therefore, modeling a production system by an order-preserving 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.

4.2 Supremal Order-preserving Language

In order to analyze properties of order-preserving languages, we introduce

$$\mathcal{L}_C^{OP}(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m) = \{L \in \mathcal{L}^{OP}(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m) \mid c(L) = C\} \quad (4.5)$$

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

Theorem 1 *Let $\mathcal{L}_C^{OP}(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m)$ be as defined in (4.5). Then, $\mathcal{L}_C^{OP}(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m)$ is closed under arbitrary union and contains a supremal element*

$$L_C^{\text{sup}}(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m) = \bigcup_{L \in \mathcal{L}_C^{OP}(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m)} L. \quad (4.6)$$

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

We first show that \mathcal{L}_C^{OP} is closed under arbitrary union. To this end, let $L_1, L_2 \in \mathcal{L}_C^{OP}$. We have to show that $L_1 \cup L_2 \in \mathcal{L}_C^{OP}$. Take an arbitrary string $s \in L_1 \cup L_2$. Then, $s \in L_1$ or $s \in L_2$. In both cases, it holds that s is order-preserving since L_1 and L_2 are both order-preserving. Hence, it follows that $L_1 \cup L_2$ is order-preserving. 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 $s' \in L_1 \subseteq L_1 \cup L_1$ such that $c(s') = C$ since $c(L_1) = C$. Together, we conclude that $L_1 \cup L_2$ is order-preserving and

$c(L_1 \cup L_2) = C$. Hence, $L_1 \cup L_2 \in \mathcal{L}_C^{\text{OP}}$. Now, consider $L_C^{\text{sup}} = \bigcup_{L \in \mathcal{L}_C^{\text{OP}}} L$. Since $\mathcal{L}_C^{\text{OP}}$ is closed under arbitrary union, it must hold that $L_C^{\text{sup}} \in \mathcal{L}_C^{\text{OP}}$. Hence, L_C^{sup} is indeed the supremal order-preserving language with capacity C .

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

$$f : \Sigma^* \rightarrow (\Sigma^{\text{in}})^* : f(s) = \text{suf}_{|p^{\text{out}}(s)|}(p^{\text{in}}(s)) \quad (4.7)$$

for any given $s \in L_C^{\text{sup}}$. That is, $f(s)$ characterizes the string of input events in s , whose corresponding output events did not occur, yet. Consider for example the string $s = \mathbf{a}_1\mathbf{a}_2\mathbf{b}_1\mathbf{b}_2\mathbf{a}_2\mathbf{b}_2\mathbf{a}_1\mathbf{a}_2 \in L_2 = L_m(G_2)$ in Fig. 4.1. Then, $f(s) = \text{suf}_{|\mathbf{b}_1\mathbf{b}_2\mathbf{b}_2|}(p^{\text{in}}(s)) = \text{suf}_3(\mathbf{a}_1\mathbf{a}_2\mathbf{a}_2\mathbf{a}_1\mathbf{a}_2) = \mathbf{a}_1\mathbf{a}_2$. Then, we define the equivalence relation \equiv_{OP} as the equivalence kernel of f such that for any two strings $s, s' \in \overline{L}_C^{\text{sup}}$,

$$s \equiv_{\text{OP}} s' \iff f(s) = f(s'). \quad (4.8)$$

Using \equiv_{OP} , we introduce the automaton $G_C^{\text{sup}} = (X_C^{\text{sup}}, \Sigma, \delta_C^{\text{sup}}, x_{0,C}^{\text{sup}}, X_{m,C}^{\text{sup}})$ such that $X = \overline{L}_C^{\text{sup}} / \equiv_{\text{OP}}$ is the quotient set of $\overline{L}_C^{\text{sup}}$, $x_{0,C}^{\text{sup}} = [\epsilon]$ and $X_{m,C}^{\text{sup}} = \{[\epsilon]\}$. Furthermore, the transition relation is defined such that for all $s \in \overline{L}_C^{\text{sup}}$ and $\sigma \in \Sigma$ with $s\sigma \in \overline{L}_C^{\text{sup}}$

$$\delta_C^{\text{sup}}([s], \sigma) = [s\sigma]. \quad (4.9)$$

Then, Theorem 2 shows that G_C^{sup} has a finite number of states and recognizes the supremal order-preserving language with capacity C . In particular, L_C^{sup} is regular.

Theorem 2 *Let L_C^{sup} be as defined in (4.5). Then, L_C^{sup} is a regular language and it holds that $|X_C^{\text{sup}}| = \sum_{i=0}^C |\Sigma^{\text{in}}|^i$.*

We show the theorem by constructing an automaton $G = (X, \Sigma, \delta, x_0, X_m)$ that is isomorphic to G_C^{sup} . We define $x_0 = [\epsilon]$, $X_m = \{[\epsilon]\}$. Furthermore, the transition relation is defined with the following rules:

$$\text{R1 } \delta([\epsilon], \alpha_1) = [\alpha_1] \text{ for all } \alpha_1 \in \Sigma^{\text{in}}$$

$$\text{R2 } \delta([\alpha_1], \omega_1) = [\epsilon] \text{ for } \omega_1 = m(\alpha_1)$$

R3 $\delta([\alpha_1, \dots, \alpha_j], \alpha_{j+1}) = [\alpha_1, \dots, \alpha_j \alpha_{j+1}]$ for all $[\alpha_1, \dots, \alpha_j] \in X$ and $\alpha_{j+1} \in \Sigma^{\text{in}}$, $j = 1, \dots, C-1$

R4 $\delta([\alpha_1, \alpha_2, \dots, \alpha_j], \omega_1) = [\alpha_2, \dots, \alpha_j]$ for all $[\alpha_1, \alpha_2, \dots, \alpha_j] \in X$ and $\omega_1 = m(\alpha_1)$, $j = 2, \dots, C$.

First, we show that G has the same number of states as G_C^{sup} . To this end, we observe that the state set X contains the equivalence classes of \equiv_{OP} . Since any string $s \in L_C^{\text{sup}}$ has a capacity $c(s) \leq C$, it must be the case that $|f(s)| = |p^{\text{in}}(s)| - |p^{\text{out}}(s)| \leq C$. Hence, the possible equivalence classes are given by $[\epsilon]$ and sequences $[\alpha_1 \cdots \alpha_j]$ for $1 \leq j \leq C$ and with arbitrary input events $\alpha_i \in \Sigma^{\text{in}}$, $1 \leq i \leq j$. That is, the state set of G_C^{sup} is $X_C^{\text{sup}} = \{[\epsilon]\} \cup \{[\alpha_1] | \alpha_1 \in \Sigma^{\text{in}}\} \cup \cdots \cup \{[\alpha_1 \cdots \alpha_{C-1}] | \alpha_1, \dots, \alpha_{C-1} \in \Sigma^{\text{in}}\} \cup \{[\alpha_1 \cdots \alpha_C] | \alpha_1, \dots, \alpha_C \in \Sigma^{\text{in}}\}$. This coincides with the state set X of G .

Considering that there are $|\Sigma^{\text{in}}|^j$ combinations for each equivalence class $[\alpha_1 \cdots \alpha_j]$, $j = 1, \dots, C$ and there is one equivalence class $[\epsilon]$, it already follows that G has a finite number of

$$|X| = \sum_{j=0}^C |\Sigma^{\text{in}}|^j \quad (4.10)$$

states. It remains to show that $L_m(G) = L_m(G_C^{\text{sup}}) = L_C^{\text{sup}}$. To this end, we show that (i) $L_m(G) \subseteq L_C^{\text{sup}}$ and (ii) $L_m(G) \supseteq L_C^{\text{sup}}$.

For (i), it is required to show that each string $s \in L_m(G)$ is order-preserving with capacity $c(s) \leq C$. We use induction on the string length to show that for each prefix $s_k := \text{pre}_k(s) \leq s$, $p^{\text{out}}(s_k) \leq m(p^{\text{in}}(s_k))$, $c(s_k) \leq C$ and $\delta(x_0, s_k) = [f(s_k)]$.

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

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

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 $\alpha_1 \in \Sigma^{\text{in}}$ according to (R1). Furthermore, it

holds that $p^{\text{out}}(s_{k+1}) = p^{\text{out}}(s_k \alpha_1) = p^{\text{out}}(s_k) \leq m(p^{\text{in}}(s_{k+1})) = m(p^{\text{in}}(s_k \alpha_1)) = m(p^{\text{in}}(s_k))m(\alpha_1)$, $c(s_{k+1}) = c(s_k \alpha_1) = \max\{c(s_k), 1\} \leq C$ and $\delta(x_0, s_{k+1}) = \delta([\epsilon], \alpha_1) = [\alpha_1] = [f(s_{k+1})]$. In the second case, let $|f(s_k)| = C$. That is, $[f(s_k)] = [\alpha_1 \alpha_2 \cdots \alpha_C]$ for some $\alpha_1, \dots, \alpha_C \in \Sigma^{\text{in}}$. Then, $s_{k+1} = s_k \omega_1$ with $\omega_1 = m(\alpha_1) \in \Sigma^{\text{out}}$ according to (R4). Noting that $f(s_k) = \alpha_1 \alpha_2 \cdots \alpha_C$, we further conclude that

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

Furthermore, $f(s_{k+1}) = f(s_k \omega_1) = \alpha_2 \cdots \alpha_C$ and accordingly $\delta(x_0, s_{k+1}) = \delta([\alpha_1 \alpha_2 \cdots \alpha_C], \omega_1) = [\alpha_2 \cdots \alpha_C] = [f(s_{k+1})]$ 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 $\delta(x_0, s_{k+1}) = \delta([\alpha_1 \cdots \alpha_j], \alpha_{j+1}) = [\alpha_1 \cdots \alpha_j \alpha_{j+1}] = [f(s_{k+1})]$. 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 $\delta(x_0, s_{k+1}) = \delta([\alpha_1 \alpha_2 \cdots \alpha_j], \omega_1) = [\alpha_2 \cdots \alpha_j] = [f(s_{k+1})]$.

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 $\delta(x_0, s) = [f(s)]$. Considering that $s \in L_m(G)$, it is further the case that $\delta(x_0, s) = [\epsilon]$. Hence, it is the case that $m(p^{\text{in}}(s)) = p^{\text{out}}(s)$. That is, s is order-preserving 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 $s_k \in L(G)$ and $\delta(x_0, s_k) = [f(s_k)]$ in order to show that $s \in L_m(G)$. For initialization, we consider $s_0 = \epsilon$. Then, it holds that $\delta(x_0, \epsilon) = [\epsilon] = [f(s_0)]$. For the induction step, we let $k < |s|$ and assume

that $\delta(x_0, s_k) = [f(s_k)]$. Then, we show that $\delta(x_0, s_{k+1}) = [f(s_{k+1})]$. 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 order-preserving, 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 $\delta([\epsilon], \alpha_1) = \delta(x_0, s_k \alpha_1) = \delta(x_0, s_{k+1}) = [\alpha_1] = [f(s_{k+1})]$. 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 order preserving, it must be the case that $\sigma = \omega_1$ for $\omega_1 = m(\alpha_1)$. But then, (R4) ensures that $\delta([\alpha_1 \alpha_2 \cdots \alpha_C], \alpha_1) = \delta(x_0, s_k \alpha_1) = \delta(x_0, s_{k+1}) = [\alpha_2 \cdots \alpha_C] = [f(s_{k+1})]$. Third, consider that $f(s_k) = \alpha_1 \alpha_2 \cdots \alpha_j$ for $1 \leq j < C$. Since $0 < |f(s_k)| < C$ and s is order preserving, 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 $\delta([\alpha_1 \cdots \alpha_j], \alpha_{j+1}) = \delta(x_0, s_k \alpha_{j+1}) = \delta(x_0, s_{k+1}) = [\alpha_1 \cdots \alpha_j \alpha_{j+1}] = [f(s_{k+1})]$. If $\sigma = \omega_1$, (R2) or (R4) ensure that $\delta([\alpha_1 \alpha_2 \cdots \alpha_j], \omega_1) = \delta(x_0, s_k \omega_1) = \delta(x_0, s_{k+1}) = [\alpha_2 \cdots \alpha_j] = [f(s_{k+1})]$.

In particular, for $k+1 = |s|$, we have that $\delta(x_0, s) = [f(s)] = [\epsilon] = x_0$ since s is order-preserving and, hence, $|p^{\text{in}}(s)| = |p^{\text{out}}(s)|$. That is, $s \in L_m(G)$.

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 (4.10) 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_{OP} as defined in (4.8) and the transition relation of G_C^{sup} is defined by the rules (R1) to (R4) in the proof of Theorem 2. It is hence straightforward to compute G_C^{sup} . As an example, consider $\Sigma = \Sigma^{\text{in}} \cup \Sigma^{\text{out}} = \{\mathbf{a}_1, \mathbf{a}_2\} \cup \{\mathbf{b}_1, \mathbf{b}_2\}$. Then, Fig. 4.1 shows $G_3 = G_1^{\text{sup}}$ and $G_4 = G_2^{\text{sup}}$.

Moreover, as a more practical example, we consider a conveyor belt (CB) in a production system 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 an order-preserving language with the input events $\Sigma^{\text{in}} = \{\mathbf{i}_{P1}, \mathbf{i}_{P2}, \mathbf{i}_{P3}\}$ and the output events $\Sigma^{\text{out}} = \{\mathbf{o}_{P1}, \mathbf{o}_{P2}, \mathbf{o}_{P3}\}$. The mapping m is defined by $m(\mathbf{i}_{Pi}) = \mathbf{o}_{Pi}$ for $i = 1, 2, 3$ and the capacity is $C = 2$. Hence, the canonical recognizer G_2^{sup} for the CB as shown in Fig. 4.2 has 13 states. Hereby, there is one product on the CB in the states shaded in light gray, whereas there

are two products on CB in the states shaded in dark gray.

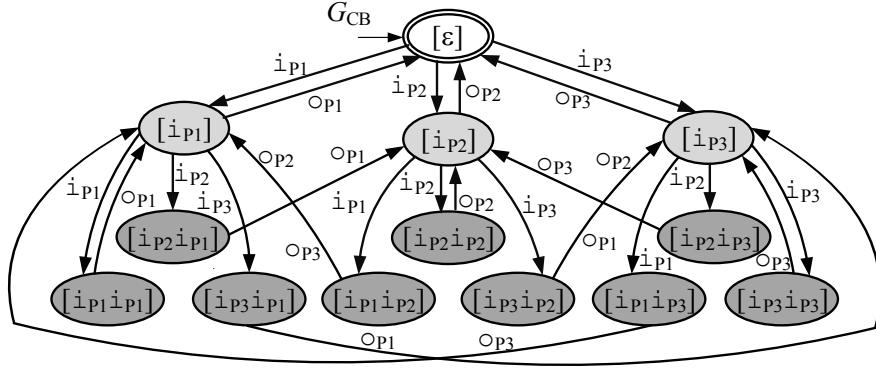


Figure 4.2: Order-preserving model of a CB with three product types P1, P2, P3 and capacity $C = 2$.

4.3 Composition of Order-preserving Languages

Practical DES are commonly composed of various modular system components. That is, we next investigate the composition of multiple components with order-preserving languages. To this end, we first consider the case of two order-preserving languages $L_i \subseteq \Sigma_i^*$ over the alphabet Σ_i with given capacities $c(L_i) = C_i$ and maps $m_i : \Sigma_i^{\text{in}} \rightarrow \Sigma_i^{\text{out}}$ 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 a production system, such languages could represent two production components, whereby the first component delivers products to the second component. 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 $p : (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma^*$, $p_i : (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma_i^*$, $i = 1, 2$ and the mapping $m : (\Sigma^{\text{in}})^* \rightarrow (\Sigma^{\text{out}})^* : m(s) = m_2(m_1(s))$, we determine the composed language

$$L = p(L_1 || L_2). \quad (4.11)$$

We next show that L is an order-preserving language with capacity $C \leq C_1 + C_2$.

Theorem 3 Assume that $L_1 \in \mathcal{L}_{C_1}^{\text{sup}}(\Sigma_1^{\text{in}}, \Sigma_1^{\text{out}}, m_1)$, $L_2 \in \mathcal{L}_{C_2}^{\text{sup}}(\Sigma_2^{\text{in}}, \Sigma_2^{\text{out}}, m_2)$, m_1 , m_2 , Σ , m and $p : (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma^*$ are as introduced above and let $\Sigma^{\text{in}} = \Sigma_1^{\text{in}}$,

$\Sigma^{\text{out}} = \Sigma_2^{\text{out}}$ and $\Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$. Further, assume that $L_1||L_2 \neq \emptyset$. Then, $L = p(L_1||L_2) \in \mathcal{L}_C^{\text{sup}}(\Sigma^{\text{in}}, \Sigma^{\text{out}}, m)$ for some $C \leq C_1 + C_2$.

In the sequel, we again write $\mathcal{L}_{C_1}^{\text{sup}}$, $\mathcal{L}_{C_2}^{\text{sup}}$ and $\mathcal{L}_C^{\text{sup}}$ for brevity.

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 order-preserving and $c(s) \leq C_1 + C_2$.

We first show that s is order-preserving. Assume the contrary. That is $s \in L$ and s is not order-preserving. Then, we can write $s = u_1\alpha u_2\omega u_3$ with $u_1, u_2, u_3 \in \Sigma^*$, $\alpha \in \Sigma^{\text{in}}$ 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 order-preserving for $i = 1, 2$, also s_i is order-preserving 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 $p_1^{\text{out}}(s'_1) = p_2^{\text{in}}(s'_2)$ 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 $c(L) = C \leq C_1 + C_2$.

We demonstrate the implications of Theorem 3 using the example automata in Fig. 4.1 and 4.3.

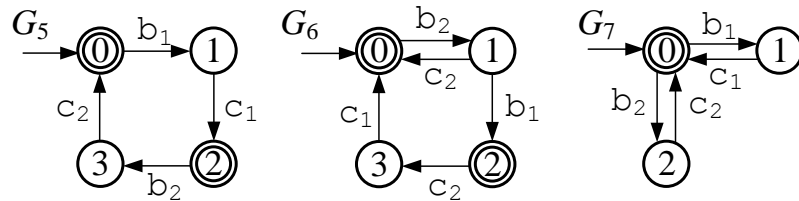


Figure 4.3: Example automata

First consider the languages $L_2 = L_m(G_2)$ with $C_2 = 2$ in Fig. 4.1 and $L_5 = L_m(G_5)$ with $C_5 = 1$ in Fig. 4.3. A canonical recognizer G_{25} (excluding the states shaded in gray) with $L_{25} = L_m(G_{25}) = p(L_2 || L_5)$ and $\Sigma^{\text{in}} = \{a_1, a_2\}$ and $\Sigma^{\text{out}} = \{c_1, c_2\}$ is shown in Fig. 4.4. It is readily observed that L_{25} is order-preserving 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 $s \in L_m(G_{25})$ is order-preserving, there might be strings in $L(G_{25})$ that cannot be extended to an order-preserving string. Next, we consider $L_2 = L_m(G_2)$ with $C_2 = 2$ in Fig. 4.1 and $L_6 = L_m(G_6)$ with $C_6 = 2$ in Fig. 4.3. A canonical recognizer G_{26} for $L_{26} = L_m(G_{26}) = p(L_2||L_6)$ with $\Sigma^{\text{in}} = \{a_1, a_2\}$ and $\Sigma^{\text{out}} = \{c_1, c_2\}$ is shown in Fig. 4.4. 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 $L_3 = L_m(G_3) = L_1^{\text{sup}}$ and $L_4 = L_m(G_4) = L_2^{\text{sup}}$ in Fig. 4.1 with $L_7 = L_m(G_7) = L_1^{\text{sup}}$ in Fig. 4.3. The resulting order-preserving languages $L_{37} = L_m(G_{37})$ and $L_{47} = L_m(G_{47})$ are shown in Fig. 4.4. 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 will formalize the observations from this example in Theorem 4 and 6.

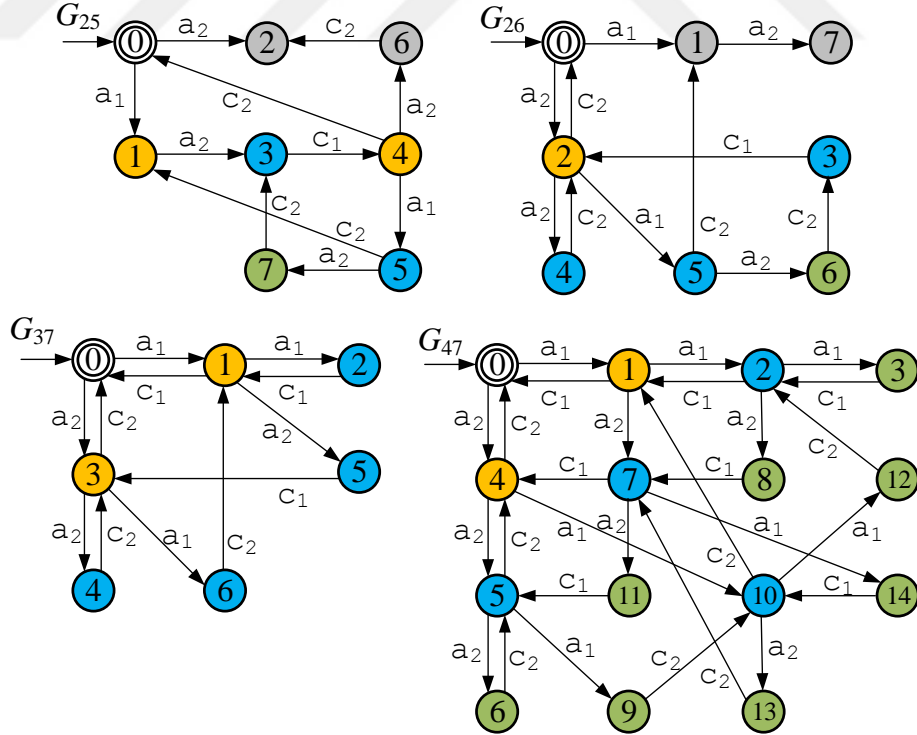


Figure 4.4: Composition of order-preserving languages.

Theorem 4 Assume that $L_1 = L_{C_1}^{sup}$ and $L_2 = L_{C_2}^{sup}$ for $C_1, C_2 > 0$. Let $m_1, m_2, m, \Sigma, p : (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma^*$, $\Sigma^{in} = \Sigma_1^{in}$, $\Sigma^{out} = \Sigma_2^{out}$ and $\Sigma_1^{out} = \Sigma_2^{in}$ be introduced as above. Then, $L = p(L_1 || L_2) = L_{C_1+C_2}^{sup}$.

That is, Theorem 4 states that the composition of two supremal order-preserving languages again yields a supremal order-preserving 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 1 determines that the capacity of L is $c(L) = C = C_1 + C_2$ if both languages L_i , $i = 1, 2$ are supremal order-preserving languages with capacity C_i .

Lemma 1 Assume that $L_1 = L_{C_1}^{sup}$ and $L_2 = L_{C_2}^{sup}$ for $C_1, C_2 > 0$. Let $p, \Sigma^{in} = \Sigma_1^{in}$, $\Sigma^{out} = \Sigma_2^{out}$ and $\Sigma_1^{out} = \Sigma_2^{in}$ be introduced as above. Then, $c(L) = C = C_1 + C_2$. We have to show that $c(L) = C = C_1 + C_2$. Since $L_1 = L_{C_1}^{sup}(\Sigma_1)$ and $L_2 = L_{C_2}^{sup}(\Sigma_2)$ 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 $C_1 \leq 2C_2$. The extension for the case $C_1 > 2C_2$ is conceptually straightforward. Since the capacity of L_1 is $c(L_1) = C_1$ and $L_1 = L_{C_1}^{sup}(\Sigma_2)$, it holds for an arbitrary $\alpha_1 \in \Sigma_1^{in}$ and $\omega_1 = m_1(\alpha_1) \in \Sigma_1^{out}$ 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 $\alpha_2 = \omega_1 \in \Sigma_2^{in} = \Sigma_1^{out}$ and $\omega_2 = m_2(\alpha_2) \in \Sigma_2^{out}$. Since the capacity of L_2 is $c(L_2) = C_2$ and $L_2 = L_{C_2}^{sup}(\Sigma_2)$, 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,

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

and

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

In the second case, $C_1 < C_2$. In addition, for simplicity, we assume that $C_2 \leq 2C_1$. The extension for the case $C_2 > 2C_1$ is conceptually straightforward. Since the capacity of L_2 is $c(L_2) = C_2$ and $L_2 = L_{C_2}^{\text{sup}}(\Sigma_2)$, 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

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

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 is full, 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

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

and

$$c(s) = |p^{\text{in}}(\underbrace{\alpha_1 \dots \alpha_1}_{C_1} \underbrace{\alpha_1 \dots \alpha_1}_{C_2 - C_1} \underbrace{\alpha_1 \dots \alpha_1}_{C_1})| - |p^{\text{out}}(\underbrace{\alpha_1 \dots \alpha_1}_{C_1} \underbrace{\alpha_1 \dots \alpha_1}_{C_2 - C_1} \underbrace{\alpha_1 \dots \alpha_1}_{C_1})| = C_1 + C_2.$$

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

Lemma 2 introduces a decomposition of $f(s)$ for any string $s \in L = p(L_1 || L_2)$.

Lemma 2 Consider $L_1, L_2, \Sigma, p, m, m_1, m_2, \Sigma^{\text{in}}$ and Σ^{out} as defined above and let $s_1 \in \bar{L}_1, s_2 \in \bar{L}_2$ and $s \in \bar{L}_1 || \bar{L}_2$ such that $s = p(s_1 || s_2)$. Further, let $f : \Sigma^* \rightarrow (\Sigma^{\text{in}})^*, f_1 : \Sigma_1^* \rightarrow (\Sigma_1^{\text{in}})^*$ and $f_2 : \Sigma_2^* \rightarrow (\Sigma_2^{\text{in}})^*$ be defined as in (4.7).

Then, it holds that

$$f(s) = m_1^{-1}(f_2(s_2)) f_1(s_1). \quad (4.12)$$

That is, assuming a string $s = p(s_1 || s_2) \in p(\bar{L}_1 || \bar{L}_2)$ the excess input events $f(s)$, whose corresponding output events did not occur, yet, are either excess input events $f_1(s_1)$ or correspond to excess input events $f_2(s_2)$. We next give the proof of Lemma 2.

We prove the lemma 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 $f(s_k) = \alpha_1 \dots \alpha_j$ for some $0 \leq j \leq C$ with $\alpha_1, \dots, \alpha_j \in \Sigma^{\text{in}}$ (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 $s_{k+1} \in p(\bar{L}_1 || \bar{L}_2)$,

$s_k = p(s_{1,k}||s_{2,k})$ 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 $s_{1,k}\alpha_{j+1} \notin \bar{L}_1$ and hence $s_{k+1} \notin p(\bar{L}_1||\bar{L}_2)$. This contradicts the assumption that $s \in p(\bar{L}_1||\bar{L}_2)$. 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 $s_{1,k}\alpha_{j+1} \in \bar{L}_1$ since $L_1 = L_{C_1}^{\text{sup}}(\Sigma_1)$. 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 $L_1 = L_{C_1}^{\text{sup}}(\Sigma_1)$, $L_2 = L_{C_2}(\Sigma_2)$ and $\gamma_{l+1} \in \Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$, it must be the case that $s_{1,k}\gamma_{l+1} \in \bar{L}_1$ and $s_{2,k}\gamma_{l+1} \in \bar{L}_2$. But then, also $s_{1,k}\gamma_{l+1}\alpha_{j+1} \in \bar{L}_1$ 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 $s_{2,k}\omega_1 \in \bar{L}_2$ since $L_2 = L_{C_2}^{\text{sup}}(\Sigma_2)$ and L_2 is order-preserving. 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 $L_1 = L_{C_1}^{\text{sup}}(\Sigma_1)$, it holds that $s_{1,k}\gamma_1 \in \bar{L}_1$ for $\gamma_1 = m_1(\alpha_1)$. Since $L_2 = L_{C_2}^{\text{sup}}(\Sigma_2)$, $\gamma_1 \in \Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$ and $\omega_1 = m_2(\gamma_1) = m(\alpha_1) \in \Sigma_2^{\text{out}}$, it is further the case that $s_{2,k}\gamma_1\omega_1 \in \bar{L}_2$. 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 $m_2^{-1}(f_2(s_{2,k+1})) = m_2^{-1}(f_2(s_{2,k}\gamma_1\omega_1)) =$

$$m_2^{-1}(f_2(s_{2,k})) = \epsilon \text{ and } f_1(s_{1,k+1}) = f_1(s_{1,k}\gamma_1) = \alpha_2 \cdots \alpha_j.$$

Since $k < |s|$ was chosen arbitrary, the proven relation also holds for $k = |s| - 1$. That is,

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

Lemma 3 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 3 *Assume that $L_1 = L_{C_1}^{\text{sup}}(\Sigma_1)$ and $L_2 = L_{C_2}^{\text{sup}}(\Sigma_2)$ for $C_1, C_2 > 0$ and $\Sigma, p : (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma^*$ are as introduced above. Further, let $\Sigma^{\text{in}} = \Sigma_1^{\text{in}}, \Sigma^{\text{out}} = \Sigma_2^{\text{out}}$ and $\Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$. Then, it holds for any $t \in \overline{L_1}||\overline{L_2}$ and $\sigma \in \Sigma$ that*

$$\begin{aligned} p(t)\sigma \in p(\overline{L_1}||\overline{L_2}) &\Rightarrow \exists v \in (\Sigma_1 \cup \Sigma_2)^* \text{ s.t.} \\ p(v) &= \sigma \text{ and } tv \in \overline{L_1}||\overline{L_2}. \end{aligned} \quad (4.13)$$

Consider an arbitrary string $t \in \overline{L_1}||\overline{L_2}$ and let $\sigma \in \Sigma$ such that $p(t)\sigma \in p(\overline{L_1}||\overline{L_2})$. We write $s = p(t)$. According to Lemma 2, 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, $f(s) = m_1^{-1}(f_2(s_2))f_1(s) = \alpha_1 \cdots \alpha_l \alpha_{l+1} \cdots \alpha_j$ 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 $L_1 = L_{C_1}^{\text{sup}}(\Sigma_1)$, 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, (4.13) is fulfilled with $v = \sigma \in (\Sigma_1 \cup \Sigma_2)^*$. 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 $L_1 = L_{C_1}^{\text{sup}}(\Sigma_1)$ and $L_2 = L_{C_2}^{\text{sup}}(\Sigma_2)$, 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

$L_1 = L_{C_1}^{\text{sup}}(\Sigma_1)$. Together, we have that $t\gamma_{l+1}\sigma \in s_1\gamma_{l+1}\sigma||s_2\gamma_{l+1} \subseteq \overline{L_1||L_2}$ and $p(t\gamma_{l+1}\sigma) = p(t)\sigma = s\sigma$. That is, (4.13) 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 $L_2 = L_{C_2}^{\text{sup}}(\Sigma_2)$, it holds that $s_2\sigma \in \overline{L_2}$. Since $\Sigma_2^{\text{in}} \cap \Sigma_1 = \emptyset$, it follows that $t\sigma \in s_1||s_2\sigma \subseteq \overline{L_1||L_2}$ and $p(t\sigma) = p(t)\sigma = s\sigma$. That is, (4.13) is fulfilled with $v = \sigma \in (\Sigma_1 \cup \Sigma_2)^*$. 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 $\gamma_1 = m_1(\alpha) \in \Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$. Since $L_1 = L_{C_1}^{\text{sup}}(\Sigma_1)$ and $L_2 = L_{C_2}^{\text{sup}}(\Sigma_2)$, 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 $L_2 = L_{C_2}^{\text{sup}}(\Sigma_2)$. Together, we have that $t\gamma_1\sigma = t\gamma_1\omega_1 \in s_1\gamma_{l+1}||s_2\gamma_{l+1}\omega_1 \subseteq \overline{L_1||L_2}$ and $p(t\gamma_{l+1}\sigma) = p(t)\sigma = s\sigma$. That is, (4.13) is fulfilled with $v = \gamma_1\sigma$.

That is, (4.13) holds in all possible cases, which proves the lemma.

Using Lemma 1 to 3, it is now possible to prove Theorem 4.

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 order-preserving and Lemma 1 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 $L \supseteq L_{C_1+C_2}^{\text{sup}}$, we take an arbitrary order-preserving string $s \in L_{C_1+C_2}^{\text{sup}}$. First, we show by induction that $s \in p(\overline{L_1||L_2})$. To this end, we write $s = \sigma_1 \cdots \sigma_{|s|}$ with $\sigma_i \in \Sigma$ for $i = 1, \dots, |s|$.

For the initialization, consider $s_0 = \text{pre}_0 s = \epsilon$. Since $\epsilon \in L_1$ and $\epsilon \in L_2$, it follows that $\epsilon \in L_1||L_2 \subseteq \overline{L_1||L_2}$. Hence, we have that $t_0 = \epsilon \in \overline{L_1||L_2}$ and $s_0 = p(t_0) \in p(\overline{L_1||L_2})$. For the induction step, we assume that $s_{k-1} = \text{pre}_{k-1} s = \sigma_1 \cdots \sigma_{k-1}$ for some $1 \leq k \leq |s|$ and there is a $t_{k-1} \in \overline{L_1||L_2}$ such that $p(t_{k-1}) = s_{k-1}$. Now consider $\sigma_k \in \Sigma$. Then, it holds that $t_{k-1} \in \overline{L_1||L_2}$ and $s_{k-1} \in p(\overline{L_1||L_2})$ and $\sigma_k \in \Sigma$ and $p(t_{k-1})\sigma_k = s_{k-1}\sigma_k = \text{pre}_k s \in p(\overline{L_1||L_2})$. According to Lemma 3, this

implies that there is a $v_k \in (\Sigma_1 \cup \Sigma_2)^*$ 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 $t = t_0v_1 \cdots v_{|s|} \in \overline{L_1}||\overline{L_2}$ and $s = \text{pre}_{|s|}s = p(t) \in p(\overline{L_1}||\overline{L_2})$.

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 $s \in L_C^{\text{sup}}$ 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 2, 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 $s \in L_C^{\text{sup}}$ was arbitrary, it follows that $L_C^{\text{sup}} \subseteq p(L_1||L_2)$, 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 order-preserving languages is non-conflicting.

Theorem 5 *Assume that $L_1 = L_{C_1}^{\text{sup}}$ and $L_2 = L_{C_2}^{\text{sup}}$ for $C_1, C_2 > 0$. Let $p : (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma^*$, $\Sigma^{\text{in}} = \Sigma_1^{\text{in}}$, $\Sigma^{\text{out}} = \Sigma_2^{\text{out}}$ and $\Sigma_1^{\text{out}} = \Sigma_2^{\text{in}}$ be introduced as above. Then, L_1 and L_2 are non-conflicting.*

We first show that L_1 and L_2 are non-conflicting. 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 $\omega_i = m(\alpha_i) \in \Sigma^{\text{out}}$ for $1 \leq i \leq j$ and define the string $v = \omega_1 \cdots \omega_l \gamma_{l+1} \omega_{l+1} \cdots \gamma_j \omega_j \in (\Sigma_1 \cup \Sigma_2)^*$ for $\gamma_i = m_1(\alpha_i) = m_2^{-1}(\omega_i)$, $i = l+1, \dots, j$. 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 $L_1 = L_{C_1}^{\text{sup}}$, it holds that $s_1v_1 \in L_1$. 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 $i = l+1, \dots, j$. Hence, also $s_2v_2 \in L_2$. Together, we have

$$tv \in (s_1||s_2)v \subseteq s_1v_1||s_2v_2 \subseteq L_1||L_2,$$

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

The implications of Theorem 4 and 6 can be observed from G_{37} and G_{47} in Fig. 4.4 and the related automata in Fig. 4.1 and 4.3. According to the previous discussion, it is the case that $L_m(G_{37}) = L_C^{\text{sup}} = L_m(G_3) || L_m(G_7) = L_{C_1 \text{sup}} || L_{C_2 \text{sup}}$ with $C_1 = 1$, $C_2 = 1$ and $C = C_1 + C_2 = 2$. Furthermore, G_{37} is nonblocking, which indicates that $L_m(G_3)$ and $L_m(G_7)$ are nonconflicting. Similarly, $L_m(G_{47}) = L_C^{\text{sup}} = L_m(G_4) || L_m(G_7) = L_{C_1 \text{sup}} || L_{C_2 \text{sup}}$ with $C_1 = 2$, $C_2 = 1$ and $C = C_1 + C_2 = 3$ and G_{47} is nonblocking. Differently, $L_m(G_{26}) \neq L_3^{\text{sup}}(\Sigma_{26})$ since $L_m(G_2) \neq L_2^{\text{sup}}$ and $L_m(G_6) \neq L_2^{\text{sup}}$. In addition, G_{26} is blocking, that is, $L_m(G_2)$ and $L_m(G_6)$ are conflicting. Here, we further note that the conditions in Theorem 4 are sufficient but not necessary. Consider for example the order-preserving languages $L_m(G_8)$ in Fig. 4.5 and $L_m(G_7)$ in Fig. 4.3. Then, it holds that $L_m(G_8) \neq L_1^{\text{sup}}$ but $L_m(G_{87}) = L_m(G_8) || L_m(G_7) = L_2^{\text{sup}}$ and $L_m(G_8)$ and $L_m(G_7)$ are nonconflicting.

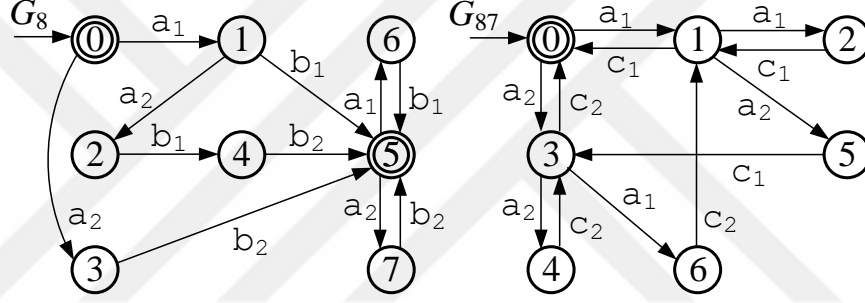


Figure 4.5: Sufficiency of Theorem 4.

An important outcome of Theorem 4 is that the composition of two supremal order-preserving languages is again a supremal order-preserving language. That is, the result of the composition operation has the same properties as its argument. Using this fact, we next state the following corollary, which shows that the composition of an arbitrary number of supremal order-preserving languages is again a supremal order-preserving language and these languages are 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 $i = 1, \dots, n$ and assume that $\Sigma_i^{\text{out}} = \Sigma_{i+1}^{\text{in}}$ for $i = 1, \dots, n-1$. Further, let $m_i : \Sigma_i^{\text{in}} \rightarrow \Sigma_i^{\text{out}}$ be defined as in Section 4.1. Assume that $L_i = L_{C_i}^{\text{sup}}(\Sigma_i^{\text{in}}, \Sigma_i^{\text{out}}, m_i)$ with $C_i > 0$ for $i = 1, \dots, n$. 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 $p : (\bigcup_{i=1}^n \Sigma_i)^* \rightarrow \Sigma^*$ as well as $m : \Sigma^{\text{in}} \rightarrow \Sigma^{\text{out}}$ such that $m(s) =$

$m_n(\dots m_1(s))$ for any $s \in (\Sigma^{in})^*$. Then, $L = p(\|_{i=1}^n L_i) = L_{C_1+\dots+C_n}^{sup}(\Sigma^{in}, \Sigma^{out}, m)$ and L_1, \dots, L_n are nonconflicting.

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

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 $L_{1,k} = L_{C_1+\dots+C_k}^{sup}$ and L_1, \dots, L_k are nonconflicting. Since $L_{1,k} = L_{C_1+\dots+C_k}^{sup}$ and $L_{k+1} = L_{C_{k+1}}^{sup}$, it directly follows from Theorem 4 that

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

With the same argument, Theorem 5 implies that $L_1 \| \cdots \| L_k$ and L_{k+1} are nonconflicting. Hence, L_1, \dots, L_{k+1} are non-conflicting.

In order to discuss the practical implication of Corollary 1, we consider a production system with an arbitrary number of production components that exchange products and that are modeled by supremal order-preserving languages. Then, it holds that the composition of the production components is again represented by the supremal order-preserving languages and the joint operation of the production components is nonblocking.

4.4 Usage of Composed Order-preserving Models in Supervisory Control

The previous section indicates that supremal order-preserving languages are nonconflicting and their composition again leads to a supremal order-preserving lan-

guage. In this section, we argue that it is possible to efficiently use composed order-preserving models for nonblocking supervisory control. As the first step, we recall the notion of a natural observer [30] that is commonly used for the supervisor synthesis based on abstracted system models [12, 13, 19].

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

$$p(s)t \in p(L) \Rightarrow \exists u \in \Sigma^* \text{ s.t. } su \in L \wedge p(u) = t.$$

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

Theorem 6 *Assume that $L_1 = L_{C_1}^{sup}$ and $L_2 = L_{C_2}^{sup}$ for $C_1, C_2 > 0$. Let $p : (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma^*$, $\Sigma^{in} = \Sigma_1^{in}$, $\Sigma^{out} = \Sigma_2^{out}$ and $\Sigma_1^{out} = \Sigma_2^{in}$ be introduced as above. Then, $p : (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma^*$ is a natural observer for $L_1||L_2$.*

In order to show that p is a natural observer for $L_1||L_2$, we take an arbitrary string $t \in \overline{L_1||L_2} = \bar{L}_1||\bar{L}_2$ and $u \in \Sigma^*$ such that $su = p(t)u \in p(L_1||L_2)$. We next show by induction that there is a $v \in (\Sigma_1 \cup \Sigma_2)^*$ 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 $k = 1, \dots, |u|$. For the initialization, we note that there is a $v_1 \in (\Sigma_1 \cup \Sigma_2)^*$ such that $tv_1 \in \bar{L}_1||\bar{L}_2$ and $p(v_1) = \sigma_1$ because of Lemma 3. For the induction step, we consider that $t_{k-1} = tv_1 \cdots v_{k-1} \in \bar{L}_1||\bar{L}_2$ such that $p(v_1 \cdots v_{k-1}) = \sigma_1 \cdots \sigma_{k-1}$. Applying Lemma 3, it directly follows that there exists a $v_k \in (\Sigma_1 \cup \Sigma_2)^*$ with $p(v_k) = \sigma_k$ and such that $tv_1 \cdots v_k \in \bar{L}_1||\bar{L}_2$ and $p(v_1 \cdots v_k) = \sigma_1 \cdots \sigma_k$. That is, for $k = |u|$, it holds for $v = v_1 \cdots v_{|u|}$ that $tv \in \bar{L}_1||\bar{L}_2$ 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, $tv \in \bar{L}_1||\bar{L}_2$ implies that $tv \in s_1||s_2$ for some $s_1 \in \bar{L}_1$, $s_2 \in \bar{L}_2$. Then, according to Lemma 2, we have that $f(s) = m_1^{-11}(f_2(s_2))f_1(s_1)$ 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.

Two main features of the observer property are used in this thesis. Consider canonical recognizers G and \hat{G} such that $L_m(G) = L$ and $L_m(\hat{G}) = p(L)$ in Definition 1. First, it is the case that \hat{G} cannot have more states (and generally has fewer states) than G if p is a natural observer for L [30]. Second, it holds that nonblocking supervisors computed using the smaller automaton \hat{G} instead of G are as well nonblocking for the original modular system with G [13, 19]. That is, when computing supervisors for modular systems, where order-preserving components are composed, it is possible to use the composed order-preserving model with a smaller state count than the composition of the original order-preserving models.

We illustrate the discussed features by the example automaton G_{47} in Fig. 4.4 that is computed such that $L_m(G_{47}) = p(L_m(G_4 || G_7))$ with G_4 in Fig. 4.1 and G_7 in Fig. 4.3. 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. 4.4 with 15 states.

We finally extend the result in Theorem 6 to the case of an arbitrary number of supremal order-preserving languages similar to Corollary 1.

Corollary 2 *Consider the setting in Corollary 1. Then, $p : (\Sigma_1 \cup \dots \cup \Sigma_n)^* \rightarrow \Sigma^*$ is a natural observer for $L_1 || L_2$.*

As the main implication of Corollary 2, it is possible to use the automaton \hat{G} with $L_m(G) = p(L_1 || \dots || L_n)$ with a smaller state count than the automaton G with $L_m(G) = L_1 || \dots || L_2$ for the nonblocking supervisor computation.

CHAPTER 5

MODELING OF MODULAR PRODUCTION SYSTEMS

Based on the discussion in the previous chapter and the results on order-preserving languages, this chapter systematically addresses the problem of modeling production systems with multiple product types and production components with a capacity greater than one. Section 5.1 formalizes the notion of an order-preserving DES model and proposes an algorithm for determining such models. Section 5.2 discusses further properties of the order-preserving model and Section 5.3 provides an illustrative FMS example.

5.1 Modular Production Systems and Order-preserving Models

We consider modular production systems (MPSs) with a set $\mathcal{C} = \{C_1, \dots, C_m\} \cup \{I, O\}$ of m production components and the virtual input component I and output component O . We assume that the MPS is processing a set of n products $\mathcal{P} = \{P_1, \dots, P_n\}$ and the main aim of this thesis is to model the flow of products in such production system. To this end, we introduce $\mathcal{P}_{C_i} \subseteq \mathcal{P}$ as the set of products that pass component C_i . Considering production components $C_i, C_j \in \mathcal{C}$ such that $C_i \neq C_j$, we define $\mathcal{P}_{C_i, C_j} \subseteq \mathcal{P}_{C_i}$ as the set of products leaving component C_i to C_j and $\mathcal{P}_{C_j, C_i} \subseteq \mathcal{P}_{C_i}$ as the set of products arriving at C_i from component C_j . In particular, $\mathcal{P}_{C_i, C_j} = \emptyset$ if no products are transported from C_i to C_j and it must hold that

$$\bigcup_{C_j \in \mathcal{C}} \mathcal{P}_{C_j, C_i} = \bigcup_{C_j \in \mathcal{C}} \mathcal{P}_{C_i, C_j}. \quad (5.1)$$

That is, all products entering component C_i should also be able to leave C_i .

For the example in Fig. 3.3, we have $\mathcal{C} = \{C, M1, M2, I, O\}$. Then, the sets \mathcal{P}_{C_i, C_j} depend on the product flow. Consider for example the scenario in Fig. 3.5 (d). Here, $\mathcal{P}_{I, C} = \{P1, P2, P3\}$, $\mathcal{P}_{C, M1} = \{P1, P3\}$, $\mathcal{P}_{C, M2} = \{P2\}$, $\mathcal{P}_{M1, O} = \{P1, P3\}$, $\mathcal{P}_{M2, O} = \{P2\}$. The remaining sets are $\mathcal{P}_{I, M1} = \mathcal{P}_{I, M2} = \mathcal{P}_{I, O} = \mathcal{P}_{O, I} = \mathcal{P}_{O, M1} = \mathcal{P}_{O, M2} = \mathcal{P}_{O, C} = \mathcal{P}_{M1, I} = \mathcal{P}_{M2, I} = \mathcal{P}_{C, I} = \emptyset$.

Using the information about the product transport between production components, we suggest to define appropriate alphabets for the DES model of each production component. Specifically, for any $C_i \in \mathcal{C}$ we introduce the input alphabet $\Sigma_{C_i}^{\text{in}}$ and the output alphabet $\Sigma_{C_i}^{\text{out}}$ as

$$\Sigma_{C_i}^{\text{in}} = \bigcup_{C_j \in \mathcal{C}} \bigcup_{P_k \in \mathcal{P}_{C_j, C_i}} C_j - C_i P_k \cup \bigcup_{P_k \in \mathcal{P}_{I, C_i}} \text{in} C_i P_k, \quad (5.2)$$

$$\Sigma_{C_i}^{\text{out}} = \bigcup_{C_j \in \mathcal{C}} \bigcup_{P_k \in \mathcal{P}_{C_i, C_j}} C_i - C_j P_k \cup \bigcup_{P_k \in \mathcal{P}_{C_i, O}} \text{out} C_i P_k. \quad (5.3)$$

In words, $\Sigma_{C_i}^{\text{in}}$ contains events for each product transport from neighboring components (including the input component I) and for each possible product type. Similarly, $\Sigma_{C_i}^{\text{out}}$ contains events for each product transport to neighboring components (including the output component O) with each possible product type. Then, the overall alphabet of any production component $C_i \in \mathcal{C}$ is $\Sigma_{C_i} = \Sigma_{C_i}^{\text{in}} \cup \Sigma_{C_i}^{\text{out}}$.

We note that the alphabets of the models in Section 3.1.2 are selected according to (5.2) and (5.3). For example, we have $\Sigma_C^{\text{in}} = \{\text{in} C_{P1}, \text{in} C_{P2}, \text{in} C_{P3}\}$ and $\Sigma_C^{\text{out}} = \{C - M1_{P1}, C - M2_{P2}, C - M1_{P3}\}$ for component C in Fig. 3.9.

Finally, we write c_{C_i} for the product capacity of component $C_i \in \mathcal{C}$ (maximum number of products that can be in component C_i simultaneously).

Using c_{C_i} , Σ_{C_i} , \mathcal{P}_{C_j, C_i} and \mathcal{P}_{C_i, C_j} for $C_j \in \mathcal{C}$, we define a general automaton model $G_{C_i} = (X_{C_i}, \Sigma_{C_i}, \delta_{C_i}, x_{0, C_i}, X_{m, C_i})$ for a production component C_i with multiple product types and a product capacity that is greater than one. Referring to G_C in Fig. 3.8 and 3.9, we note that each state of the automaton model represents a possible sequence of products entering the component. For a production compo-

nent C_i , with the capacity c_{C_i} , the state set X_{C_i} is hence given by

$$X_{C_i} = \{E\} \cup \bigcup_{l=1}^{c_{C_i}} \binom{l}{k=1} \mathcal{P}_{C_i}. \quad (5.4)$$

Hereby, E is the empty state and the Cartesian product with the set \mathcal{P}_{C_i} represents the combinations of products in the component. Then, the initial state is given as $x_{0,C_i} = E$ and the set of marked states is $X_{m,C_i} = \{E\}$ since it is always desired to go back to the empty state of a production component in order to complete all production tasks.

For example, the state set for component C in Fig. 3.8 with the products in $\mathcal{P}_C = \{P1, P2\}$ is found as $X_C = \{E, P1, P2, (P1, P1), (P1, P2), (P2, P1), (P2, P2)\}$. Then, $x_{0,C} = E$ and $X_{m,C} = \{E\}$.

It remains to determine the transition relation δ_{C_i} . To this end, we first observe that any input event in $\Sigma_{C_i}^{\text{in}}$ adds one new product to the production component and each output event in $\Sigma_{C_i}^{\text{out}}$ removes the oldest product from the production component. That is, assuming that the current state of C_i is (Pk, \dots, Pl) and an input event with product Pm occurs, the new state is (Pm, Pk, \dots, Pl) (adding the new product PM). Similarly, assuming that the current state of C_i is (Pk, \dots, Pl, Pm) , then only output events with product Pm are possible and the new state is (Pk, \dots, Pl) (taking out the "oldest" product Pm). Using this observation, the transition relation is defined for any state $x \in X_{C_i}$ and $\sigma_{Pm} \in \Sigma_{C_i}$ as

$$\delta_{C_i}(x, \sigma_{Pm}) = \begin{cases} (Pm) & \text{if } x = E \text{ and } \sigma_{Pm} \in \Sigma_{C_i}^{\text{in}} \\ (Pm, Pk, \dots, Pl) & \text{if } x = (Pk, \dots, Pl) \text{ and } \sigma_{Pm} \in \Sigma_{C_i}^{\text{in}} \\ E & \text{if } x = (Pm) \text{ and } \sigma_{Pm} \in \Sigma_{C_i}^{\text{out}} \\ (Pk, \dots, Pl) & \text{if } x = (Pk, \dots, Pl, Pm) \text{ and } \sigma_{Pm} \in \Sigma_{C_i}^{\text{out}} \\ \text{undefined} & \text{otherwise.} \end{cases} \quad (5.5)$$

We note that this model coincides with the automaton model for a supremal order-preserving language as introduced in Section 4.2 and formulated in the proof of Theorem 2.

Consider for example the model G_C in Fig. 3.8. Here, the transitions $\delta_C(E, \text{in}C_{P1}) =$

$(P1)$, $\delta_C((P1), \text{in}C_{P2}) = (P2, P1)$, $\delta_C((P1, \text{out}C_{P1}) = E$ and $\delta_C((P1, P2), \text{out}C_{P2}) = (P1)$ are introduced according to the rules in (5.5).

5.2 Order-preserving Models

We recall that the main objective of this thesis is the modeling of production components that can hold multiple products with different types, while preserving the order to products entering and leaving the component. For the first time in the literature, the model introduced in the proof of Theorem 2 and restated for the special case of MPC in Section 5.1 fulfills this purpose. In particular, we point out that the label of each state in G_{C_i} keeps track of the order of products entering the component C_i . In the list $[Pk, \dots, Pl]$, old products appear on the right, whereas new products appear on the left. On the one hand, any new product P_m entering the component with an input event $\sigma_{P_m} \in \Sigma_{C_i}^{\text{in}}$ is added to the state label from the left, to obtain the next state (P_m, Pk, \dots, Pl) in (5.5). Hence, indeed, the state label always correctly characterizes the order of incoming products. On the other hand, any product that leaves the system at some state (Pk, \dots, Pl, P_m) must be the oldest product P_m on the right of the tuple as stated in (5.5). Hence, products indeed leave the component in the same order as entering the component. Accordingly, we denote the model in Section 5.1 as an *order-preserving* DES model since its marked language is the supremal order-preserving language for the respective alphabet.

As a further interesting feature of the proposed model, it is possible to determine the number of states $|X_{C_i}|$ depending on the product capacity c_{C_i} and the number of products $|\mathcal{P}_{C_i}|$. Inspecting (5.4) and the proof of Theorem 2, it holds that

$$|X_{C_i}| = 1 + \sum_{k=1}^{c_{C_i}} |\mathcal{P}_{C_i}|^k = \sum_{k=0}^{c_{C_i}} |\mathcal{P}_{C_i}|^k. \quad (5.6)$$

In particular, for each possible number k of products in the component with $0 \leq k \leq n_{C_i}$, there are $|\mathcal{P}_{C_i}|^k$ combinations of product types.

For example, an order-preserving model for a production component C_i with capacity $n_{C_i} = 2$ and $|\mathcal{P}_{C_i}| = 3$ product types, the model has $1 + 3 + 9 = 13$ states

(see Fig. 3.9) and the model of a production component with capacity $n_{C_i} = 4$ and $|\mathcal{P}_{C_i}| = 2$ product types has $1 + 2 + 2^2 + 2^3 + 2^4 = 31$ states.

5.3 Illustrative Example

In this section, we apply the modeling technique introduced in Section 5.1 to an FMS example that is adapted from [8, 15]. Section 5.3.1 describes the components of the example system and Section 5.3.2 determines suitable automata models. The supervisor computation for the FMS is discussed in Section 5.3.3.

5.3.1 FMS Example

The FMS example consists of 3 robots R1, R2, R3 and two machines M1 and M2. That is, $\mathcal{C} = \{R1, R2, R3, M1, M2, I, O\}$ including the virtual input and output components. Hereby, it is assumed that the capacity of the robots is $c_{R1} = c_{R2} = c_{R3} = 1$ whereas the machines can hold up to $c_{M1} = c_{M2} = 2$ products. There are two product types P1 and P2 that arrive at the FMS from the virtual input component I and leave the FMS to the virtual output component O. P1 is picked from I by R3, moved to M2, then transported to M1 by R2 and finally transported to O by R1. P2 is picked from I by R1, moved to M1, then transported to M2 by R2 and finally transported to O by R3. That is, both product paths overlap and both products are processed by M1 and M2, whose capacity is two. Hence, the models for these machines need to keep track of the product order.

5.3.2 Order-preserving Model for the Example System

In order to model the FMS, we first determine the sets \mathcal{P}_{C_i, C_j} that characterize the product exchange between neighboring components. It holds that $\mathcal{P}_{I, R3} = \{P1\}$, $\mathcal{P}_{R3, M2} = \{P1\}$, $\mathcal{P}_{M2, R2} = \{P1\}$, $\mathcal{P}_{R2, M1} = \{P1\}$, $\mathcal{P}_{M1, R1} = \{P1\}$, $\mathcal{P}_{R1, O} = \{P1\}$, $\mathcal{P}_{I, R1} = \{P2\}$, $\mathcal{P}_{R1, M1} = \{P2\}$, $\mathcal{P}_{M1, R2} = \{P2\}$, $\mathcal{P}_{R2, M2} = \{P2\}$, $\mathcal{P}_{M2, R3} = \{P2\}$, $\mathcal{P}_{R3, O} = \{P2\}$. All remaining sets are empty.

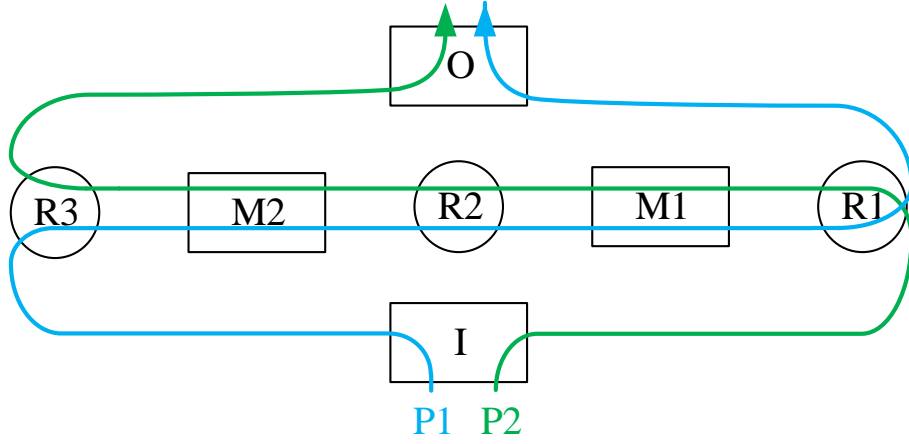


Figure 5.1: Schematic of the FMS.

Accordingly, it is possible to define the input and output alphabets of different production components according to (5.2) and (5.3) as $\Sigma_{R1}^{in} = \{inR1_{P2}, M1-R1_{P1}\}$, $\Sigma_{R1}^{out} = \{R1-M1_{P2}, outR1_{P1}\}$, $\Sigma_{R2}^{in} = \{M2-R2_{P1}, M1-R2_{P2}\}$, $\Sigma_{R2}^{out} = \{R2-M1_{P1}, R2-M2_{P2}\}$, $\Sigma_{R3}^{in} = \{inR3_{P1}, M2-R3_{P2}\}$, $\Sigma_{R3}^{out} = \{R3-M2_{P1}, outR3_{P2}\}$, $\Sigma_{M1}^{in} = \{R2-M1_{P1}, R1-M1_{P2}\}$, $\Sigma_{M1}^{out} = \{M1-R1_{P1}, M1-R2_{P2}\}$ and $\Sigma_{M2}^{in} = \{R3-M2_{P1}, R2-M2_{P2}\}$, $\Sigma_{M2}^{out} = \{R2-R2_{P1}, M2-R3_{P2}\}$.

Considering that the robots all have a capacity of one product, their models according to (5.4) and (5.5) have 3 states as shown in Fig. 5.2.

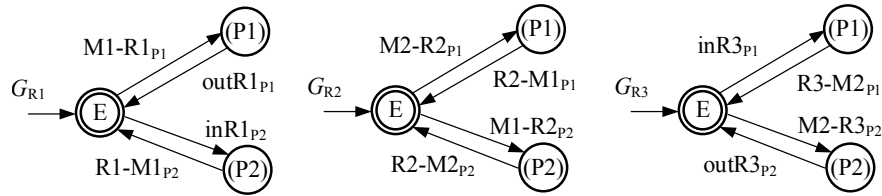


Figure 5.2: Robot models.

Differently, the machines have a capacity of two product and two product types pass M1 and M2. Hence, their models need to remember the product order similar to the model in Fig. 3.8. The resulting models for M1 and M2 are shown in Fig. 5.3 and 5.4, respectively.

The overall model G of the FMS is then given by the synchronous composition

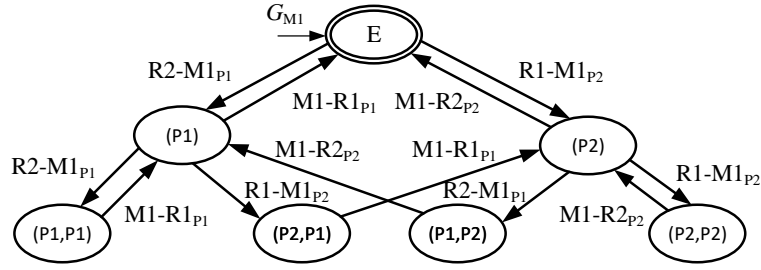


Figure 5.3: Model of machine M1.

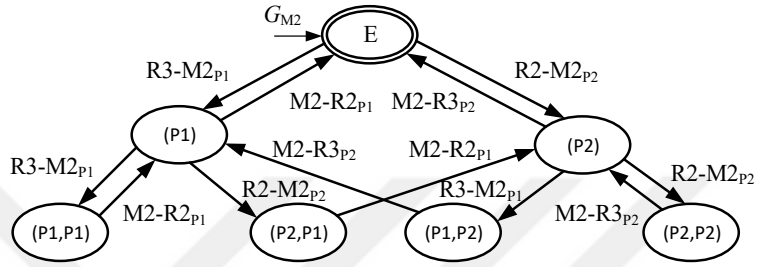


Figure 5.4: Model of machine M2.

of the component models are

$$G = G_{R1} || G_{R2} || G_{R3} || G_{M1} || G_{M2}. \quad (5.7)$$

Since this model has 2368 states, it is not displayed in the thesis.

5.3.3 Supervisor Computation for the Example System

The FMS model G already represents all the possible product paths as specified in Fig. 5.1. However, it turns out that G is a blocking automaton, that is, the uncontrolled FMS will encounter deadlock situations. Two examples of such deadlock situation are illustrated in Fig. 5.5. The figure shows the production components (robots and machines), whereby, the machines are displayed in the form of a FIFO (first-in-first-out) queue that can hold up to two products and such that the product entering the component first will also leave first. Products of type P1 are represented by blue disks, whereas products of type P2 are shown as green triangles.

Consider the scenario on the left-hand side of Fig. 5.5. Here, two products of type P1 are present in M1 and one product of type P2 is present in R1. That is, the products in M1 need to be picked by R1, whereas the product in R1 must be placed in M1. This is not possible since both M1 and R1 are fully occupied. Hence, the FMS will deadlock when reaching this scenario. Similarly, the FMS deadlocks in the scenario on the right-hand side of Fig. 5.5. Here, M2 is fully occupied and the first product of type P2 has to be picked by R3. However, R3 is already occupied by a product of type P1, which has to move to M2.

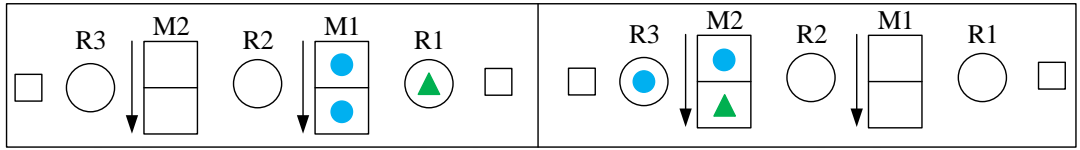


Figure 5.5: Two examples of deadlock situations.

In order to avoid such deadlock situations, we design a maximally permissive and nonblocking supervisor as described in Section 2.2. The supervisor automaton S for this case has 72 states and restricts the behavior of G in order to avoid deadlocks. Since the supervisor S is too large it cannot be displayed in this thesis.

We next illustrate the operation of the computed supervisor S and demonstrate the practicability of the proposed modeling technique. To this end, we compare the designed supervisor with a supervisor that is computed for a plant model without keeping track of the product order. In the modified plant, the robot models in Fig. 5.2 remain the same, whereas the machine models are replaced by the automata \hat{G}_{M1} and \hat{G}_{M2} in Fig. 5.6. Here, both products can leave the respective production component in the state (P1, P2) independent of the arrival order of the products. In that case, the modified plant $\hat{G} = G_{R1} || G_{R2} || G_{R3} || G_{M1} || G_{M2}$ has 24956 states and the corresponding maximally permissive and nonblocking supervisor \hat{S} has 564 states.

We next compare the operation of the supervisor S for the order-preserving model G and the supervisor \hat{S} for the modified model \hat{G} by following an example product path. Here, the left-hand side of Fig. 5.7 shows the operation

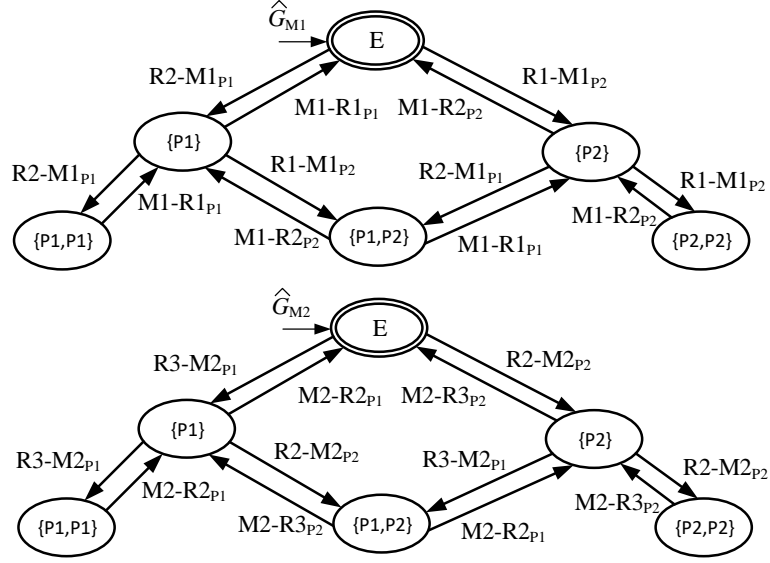


Figure 5.6: Model of the machines M1 and M2 without preserving the product order.

of S and the right-hand side shows the operation of \hat{S} . In both cases, we consider two products of type P1 and P2. The product of type P2 step by step approaches M1 and enters this machine first. The product of type P1 also moves towards M1 and enters this machine after the other product. In Fig. 5.7, this state of the FMS is reached at the step that is shaded in gray. After this step, both supervisors may exhibit a different operation. Since S is order-preserving, it must be the case that the product of type P2 which entered M1 first must also leave M1 first. That is, this product next moves to R2. After that, the product of type P1 can move to R1 and leave the FMS. Differently, the product of type P1 can move to R1 first and leave the FMS if \hat{S} is used since \hat{S} does not keep track of the product order. Re-visiting the discussion in Section 5.2, we further note that the realization of the supervisor \hat{S} needs additional information compared to the realization of S . In the described situation with two products in M1, \hat{S} needs to know which product leaves M1 first. This is only possible by installing a sensor for identifying products leaving M1. No such sensor is required when using the proposed order-preserving supervisor S .

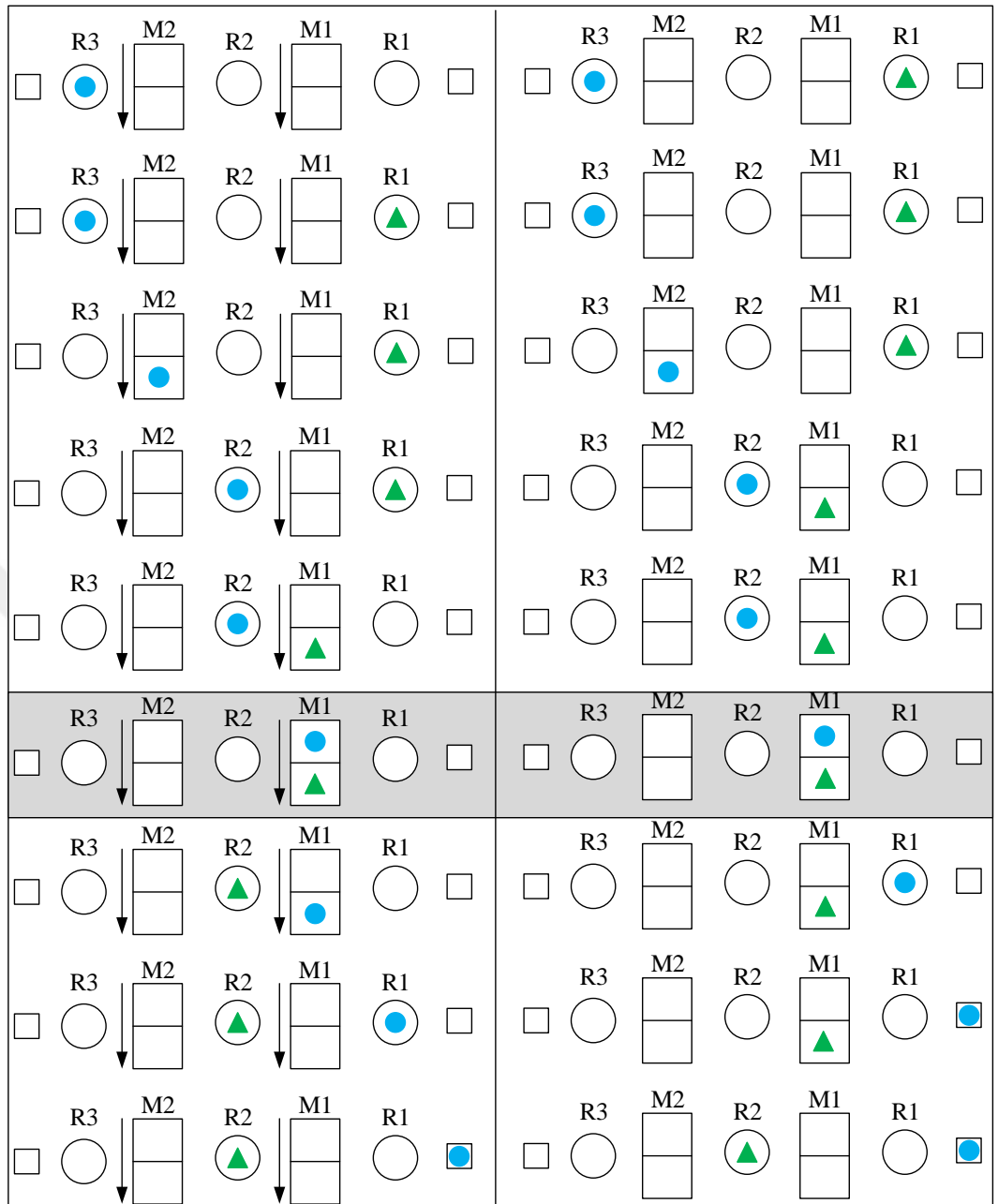


Figure 5.7: Example product path: Order-preserving case (left) and case with arbitrary product order (right).

CHAPTER 6

APPLICATION EXAMPLES

This chapter applies the order-preserving modeling technique introduced in the previous chapters to different example systems. Section 6.1 considers an FMS with multiple robots, machines and three product types. In addition, FMS whose behavior can be validated in a simulation environment are taken into account in Section 6.2. Hereby, Section 6.2 focuses on changes in the capacity of production components.

6.1 Flexible Manufacturing Systems Example

In this section, we apply the proposed modeling framework to the FMS example in [8, 15]. This example is chosen since it offers different product types that share machines and robots. We first describe the original FMS in Section 6.1.1 and then provide order-preserving models of the relevant system components and a supervisor design in Section 6.1.2. Finally, Section 6.1.3 performs a modification of the FMS that illustrates the advantage of composing order-preserving models.

6.1.1 FMS Description

The outline of the FMS in [8, 15] is shown in Fig. 6.1. 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.

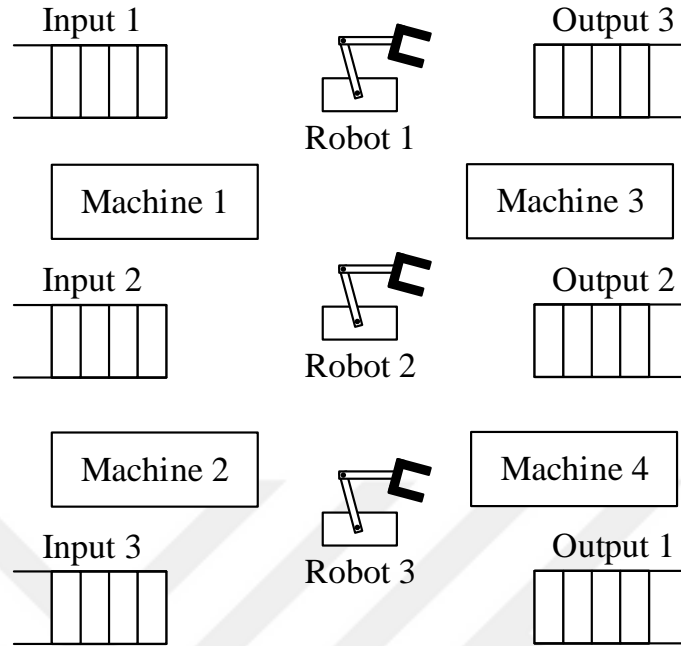


Figure 6.1: FMS overview.

In principle, it is desired to process 3 different product types using the FMS in Fig. 6.1. For the first product type, R1 takes a product from I1 and moves it to M2. Then, R2 delivers the product from M2 to O1. The second product is taken from I2 by R3 and transported to M4. Then, R2 moves the product to M3 and R1 delivers the product to O2. The third product 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. The product paths are also indicated below.

- P1: $I1 \rightarrow R2 \rightarrow M2 \rightarrow R2 \rightarrow O1$,
- P2: $I2 \rightarrow R3 \rightarrow M4 \rightarrow R2 \rightarrow M3 \rightarrow R1 \rightarrow O2$,
- P3: $I3 \rightarrow R1 \rightarrow M3 \rightarrow R2 \rightarrow M4 \rightarrow R3 \rightarrow O3$ or
 $I3 \rightarrow R1 \rightarrow M1 \rightarrow R2 \rightarrow M2 \rightarrow R3 \rightarrow O3$.

6.1.2 Modeling of the FMS

According to the FMS setup, it is clear that several production components 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. That is, it is clear that order-preserving models are needed for R1, R2, R3, M2, M3 and M4. Following the description in [8], the robots can hold a single product, whereas the machines are able to hold up to two products simultaneously. That is, a simple order-preserving model as in Fig. 6.2 is suitable to represent the robots. Hereby, we use the following convention for the event names: each event name captures the component name, where the product comes from, the component name 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 $I3-R1_{P3}$.

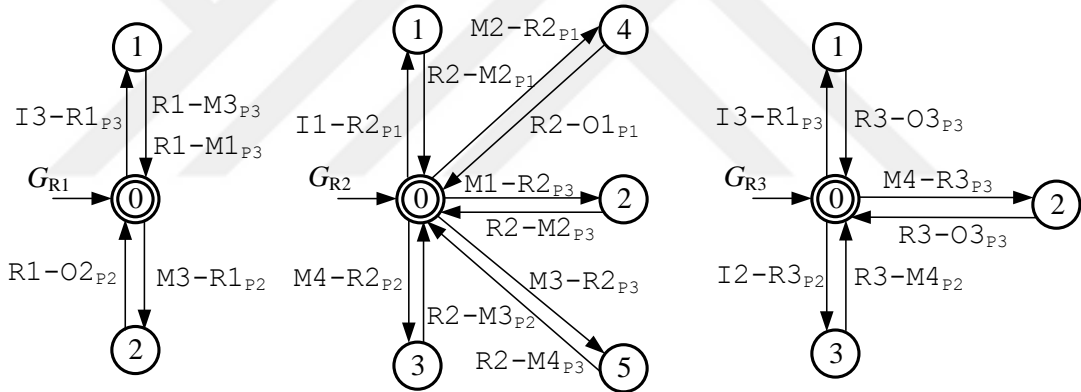


Figure 6.2: Robot models.

Different from the robots, an order-preserving model with capacity 2 is needed to capture the behavior of M2, M3 and M4 as is shown in Fig. 6.3. Since there are two different product types on each of these machines, the resulting model has 7 states. Note that no order-preserving model is needed for M1 since this machine only processes a single product type.

Finally, [8] 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

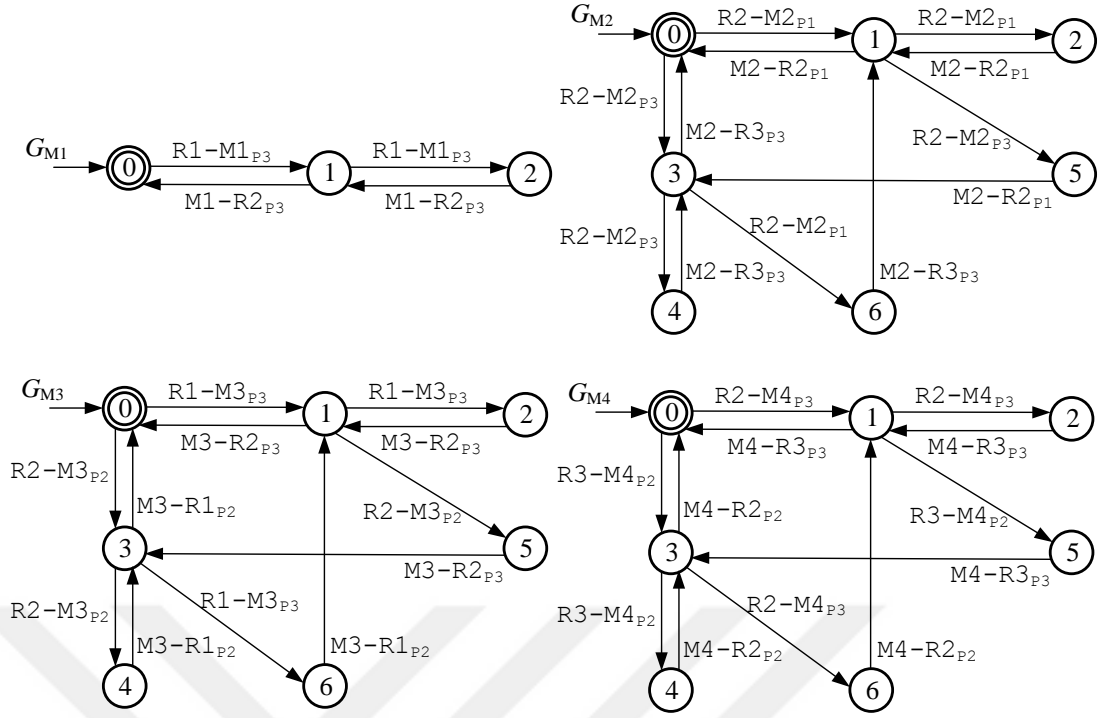


Figure 6.3: Order-preserving machine models.

and 11 for P3. These buffer models are shown in Fig. 6.4.

In this context, we note that [8] does not employ order-preserving models for M2, M3, M4. The respective models in [8] are shown in Fig. 6.5. 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 required 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 order-preserving models in Fig. 6.5 naturally keep track of the products entering and leaving each machine.

We next discuss the supervisor computation for the FMS according to Section 2.2. To this end, we first note that the aim of this thesis is not an efficient supervisor design method but the usage of order-preserving models as introduced in Chapter 5. In addition, it holds that the product paths are already encoded in

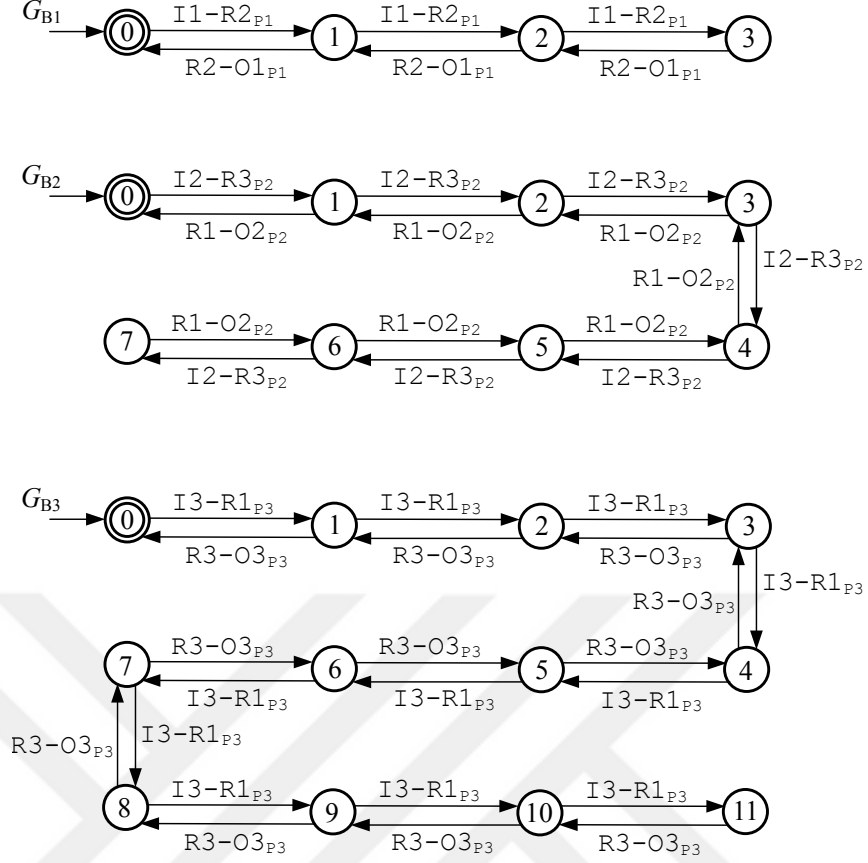


Figure 6.4: Buffer models.

the component models such that no additional specification is required. That is, we compute the overall plant as

$$G = G_{R1} || G_{R2} || G_{R3} || G_{M1} || G_{M2} || G_{M3} || G_{M4} || G_{B1} || G_{B2} || G_{B3}. \quad (6.1)$$

Then, we compute a nonblocking supervisor S such that $L_m(S||G) = \text{SupC}(L_m(G), G, \Sigma_u)$ with $\Sigma_u = \emptyset$ with 30 692 states.

In addition, we recall that the machine models in [8] as shown in Fig. 6.5 are not order-preserving. For comparison, we also compute the nonblocking supervisor \tilde{S} for the original model

$$\hat{G} = G_{R1} || G_{R2} || G_{R3} || \tilde{G}_{M1} || \tilde{G}_{M2} || \tilde{G}_{M3} || \tilde{G}_{M4} || G_{B1} || G_{B2} || G_{B3}. \quad (6.2)$$

with 27 770 states. In addition, it can be verified that $L_m(S||G) \subseteq L_m(\hat{S}||\hat{G})$. This is due to the fact that the machine models in [8] do not preserve the order

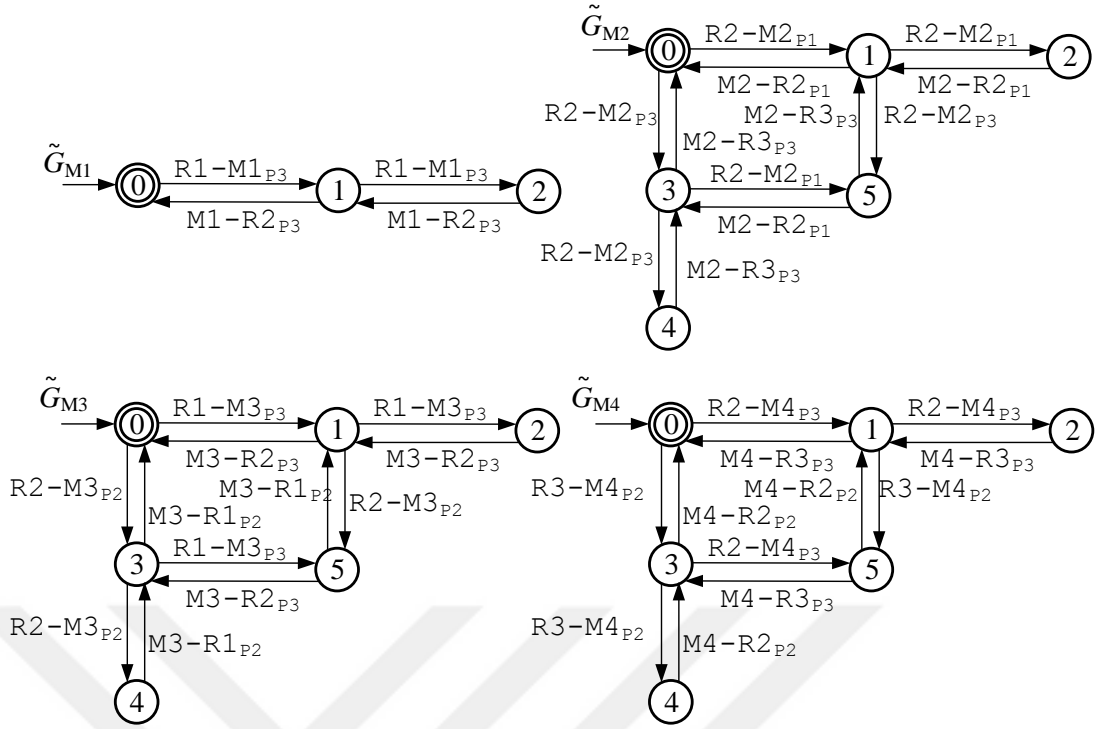


Figure 6.5: Machine models.

of products. That is, the allow products to enter and leave the machines in an arbitrary order, which is observed as additional behavior in $L_m(\hat{S}||\hat{G})$.

6.1.3 Composed Order-preserving Models for the FMS

In order to illustrate the benefits of composing order-preserving models as described in Section 4.4, we consider a slightly modified version of the FMS in Section 6.1.2. To this end, we replace the machines M2, M3, M4 by corresponding workcells W2, W3, W4 that consist of two conveyor belts (CBs) and one machine. The outline of these workcells is shown in Fig. 6.6.

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 workcell components, we use the following order-preserving models in Fig. 6.7.

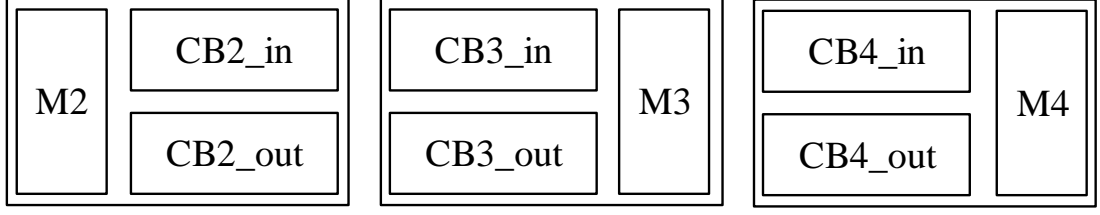


Figure 6.6: Workcell outline.

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

$$G_{W2} = G_{CB2i} || G_{M2} || G_{CB2o},$$

$$G_{W3} = G_{CB3i} || G_{M3} || G_{CB3o},$$

$$G_{W4} = G_{CB4i} || G_{M4} || G_{CB4o}$$

for the original workcell models with the alphabets $\Sigma_{W2} = \Sigma_{CB2i} \cup \Sigma_{M2} \cup \Sigma_{CB2o}$, $\Sigma_{W3} = \Sigma_{CB3i} \cup \Sigma_{M3} \cup \Sigma_{CB3o}$ and $\Sigma_{W4} = \Sigma_{CB4i} \cup \Sigma_{M4} \cup \Sigma_{CB4o}$. According to Section 4.4, we also introduce the alphabets of the reduced order-preserving models for the workcells as $\hat{\Sigma}_{W2} = \Sigma_{W2} \setminus \Sigma_{M2}$, $\hat{\Sigma}_{W3} = \Sigma_{W3} \setminus \Sigma_{M3}$ and $\hat{\Sigma}_{W4} = \Sigma_{W4} \setminus \Sigma_{M4}$ with the related natural projections $p_{W2} : \Sigma_{W2}^* \rightarrow \hat{\Sigma}_{W2}^*$, $p_{W3} : \Sigma_{W3}^* \rightarrow \hat{\Sigma}_{W3}^*$ and $p_{W4} : \Sigma_{W4}^* \rightarrow \hat{\Sigma}_{W4}^*$. That is, the reduced models only contain the external events of the workcells that are shared with the robots. Then, we write \hat{G}_{W2} , \hat{G}_{W3} and \hat{G}_{W4} for the reduced order-preserving models such that $L_m(\hat{G}_{W2}) = p_{W2}(L_m(G_{W2}))$, $L_m(\hat{G}_{W3}) = p_{W3}(L_m(G_{W3}))$ and $L_m(\hat{G}_{W4}) = p_{W4}(L_m(G_{W4}))$.

It holds that G_{W2} , G_{W3} and G_{W4} have 27 states, whereas \hat{G}_{W2} , \hat{G}_{W3} and \hat{G}_{W4} have only 15 states (which complies with (4.10)). Accordingly, a nonblocking supervisor for the overall system with the original workcell models G_{W2} , G_{W3} and G_{W4} has 2 444 288 states, whereas a nonblocking supervisor for the overall system with the reduced workcell models \hat{G}_{W2} , \hat{G}_{W3} and \hat{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 order-preserving languages.

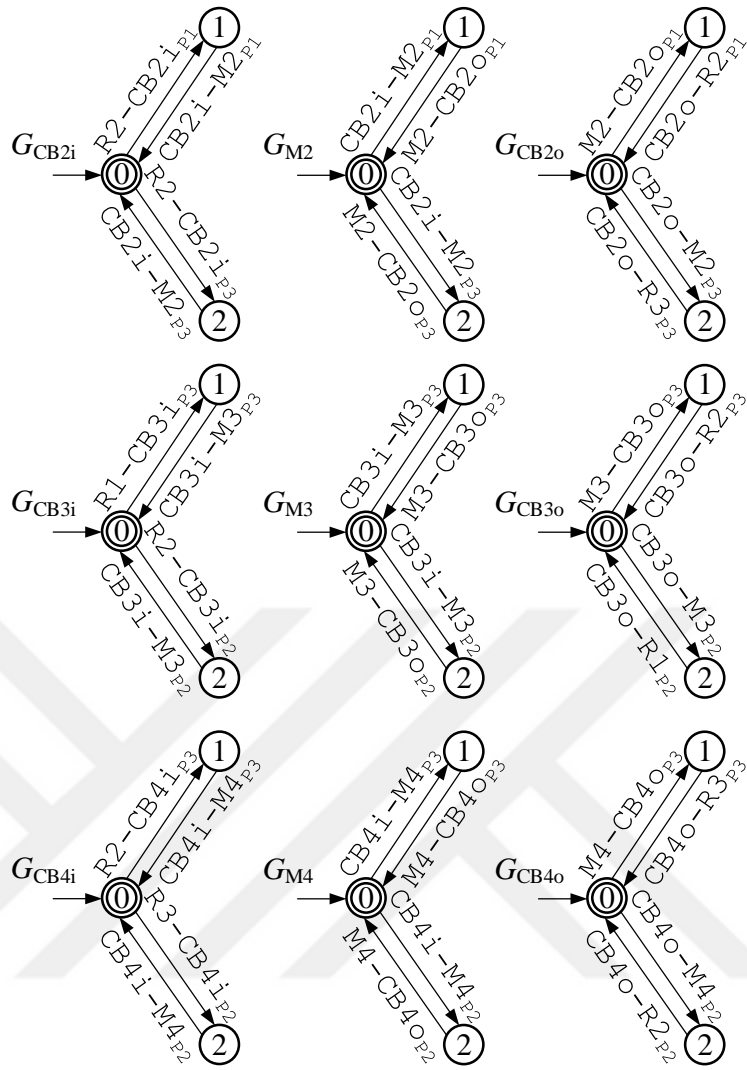


Figure 6.7: Workcell models.

6.2 Simulation of Flexible Manufacturing Systems

In this section, we investigate FMS models that can be simulated using the software library libfaudes and the manufacturing system simulator FlexFact [31, 32]. Section 6.2.1 considers a small FMS and Section 6.2.2 extends the small FMS by a long conveyor belt.

6.2.1 Small FMS Model

In this section, we consider the FMS in Fig. 6.8. It consists of two stack feeders (SF1 and SF2), two machines (M1 and M2), one rotary table (RT1) and two exit slides (XS1). The stack feeders are production components that can hold a certain number of products and then deliver these products to the FMS. In this example, we assume that SF1 and SF2 hold products of type P1 and P2, respectively. The machines process the products and deliver them to RT1. RT1 is a rotary table, that is, RT1 can rotate and hence deliver products to 4 different directions. The exit slides are simply storage areas that receive products from RT1.

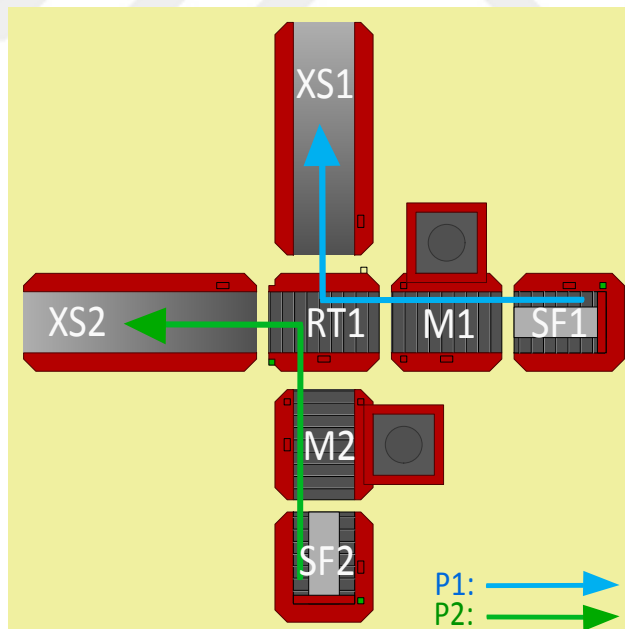


Figure 6.8: FMS model with different production components.

We next present order-preserving automata models for the small FMS. Fig. 6.9

shows the machine models G_{M1} and G_{M2} of M1 and M2. In particular, each machine receives products from the neighboring stack feeder and delivers them to RT1. Here, the event M1-wpar and M2-wpar characterize the arrival of a product at the machine. In addition, Fig. 6.9 shows models for the stack feeders and the exit slides.

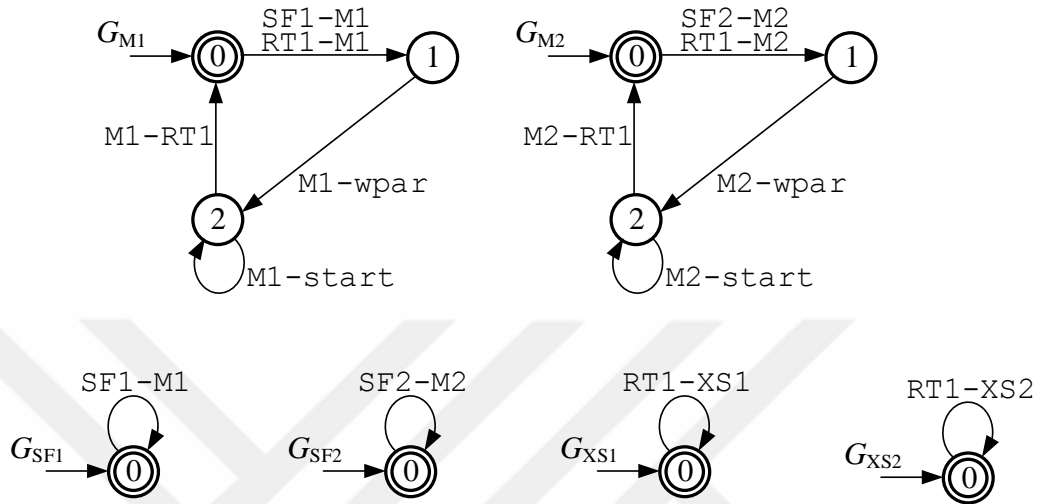


Figure 6.9: Machine, stack feeder and exit slide models of the small FMS.

The rotary table model G_{RT1} is given in Fig. 6.10. It captures that products can arrive at RT1 from M1 or M2. In addition, the model of RT1 captures that RT1 has to rotate clockwise (cw) or counter-clockwise (ccw) in order to deliver products to different directions.

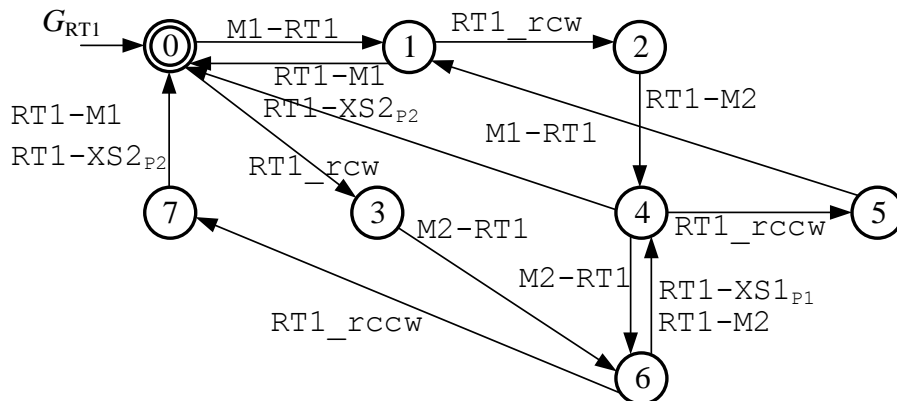


Figure 6.10: Rotary table model of the small FMS.

The overall model of the small FMS is given by the synchronous composition

$$G_{\text{FMS}} = G_{\text{SF1}} || G_{\text{M1}} || G_{\text{SF2}} || G_{\text{M2}} || G_{\text{RT1}} || G_{\text{XS1}} || G_{\text{XS2}}. \quad (6.3)$$

Different from the previous example, we next use an additional feature of order-preserving languages. Instead of using order-preserving languages for only modeling DES, we specify the desired behavior of the FMS by the order-preserving language G_{C1} in Fig. 6.11. This specification indicates that products arriving at M1 and M2 should move to RT1 in the order of arrival.

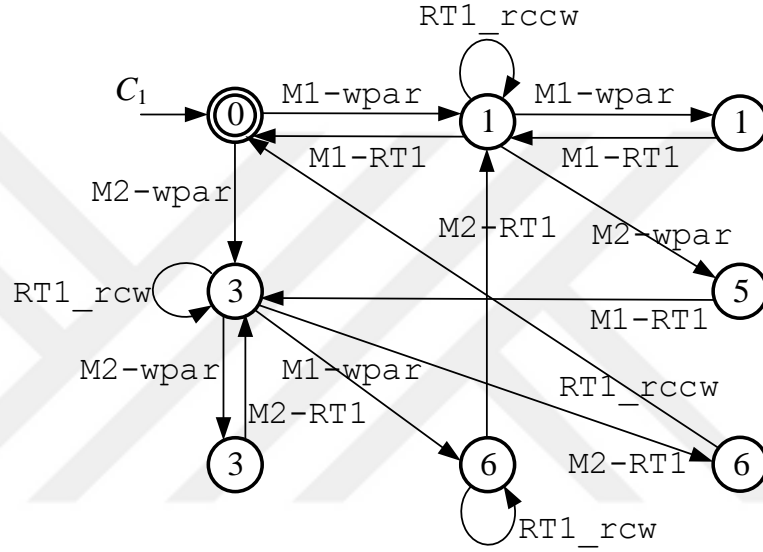


Figure 6.11: Order-preserving specification for the small FMS.

Using G_{FMS} in (6.3) and G_{C1} , it is now possible to compute a maximally permissive supervisor S_{FMS} for the FMS such that

$$L_m(S || G) = \text{supC}(L_m(G_{C1}), G_{\text{FMS}}, \emptyset). \quad (6.4)$$

For this example, S has 52 states and is hence too large to be shown in the thesis. Nevertheless, we are next able to illustrate the operation of S by showing the sequence of products in Fig. 6.12 and 6.13.

It is readily observed that the product keeps the order of entering M1 and M2.

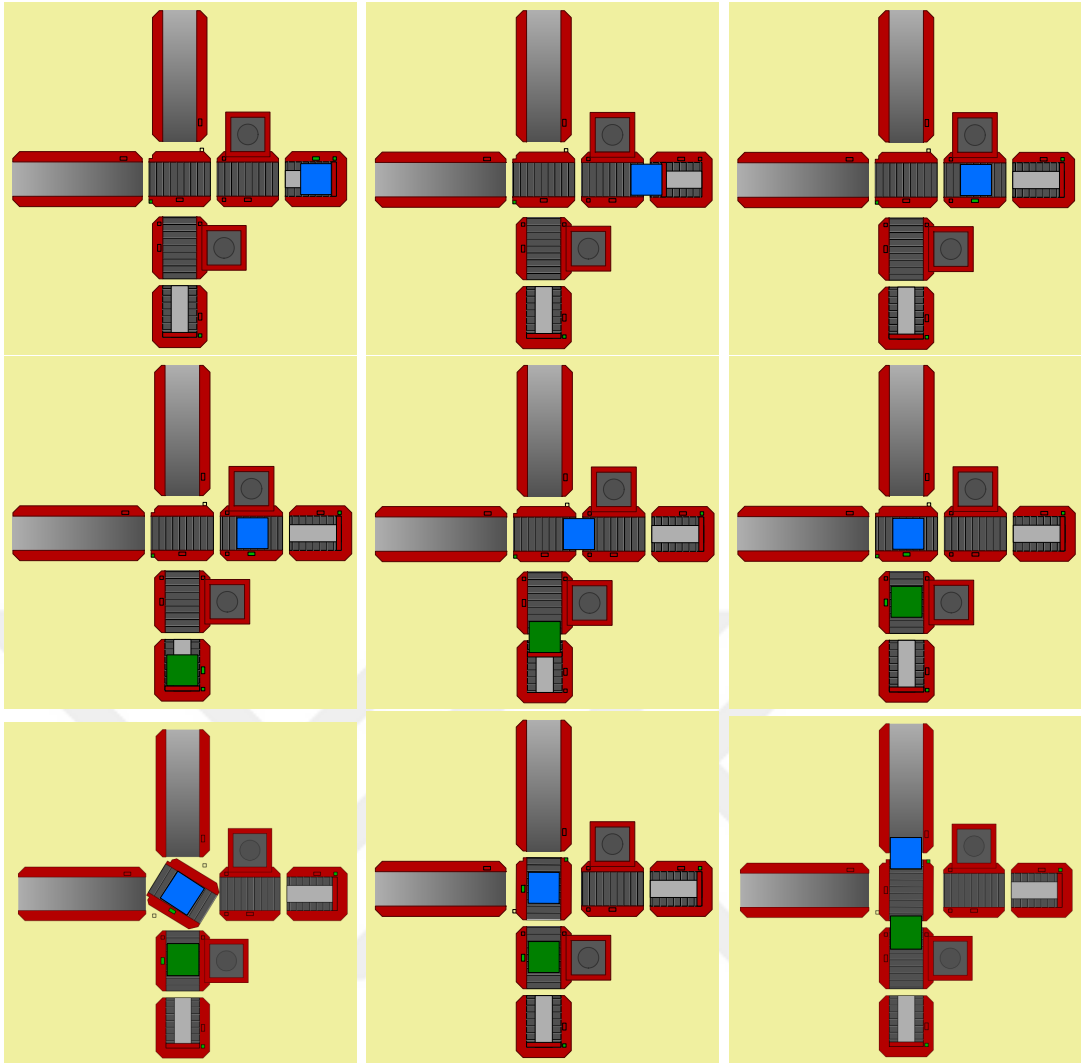


Figure 6.12: Snapshots of the FMS operation.

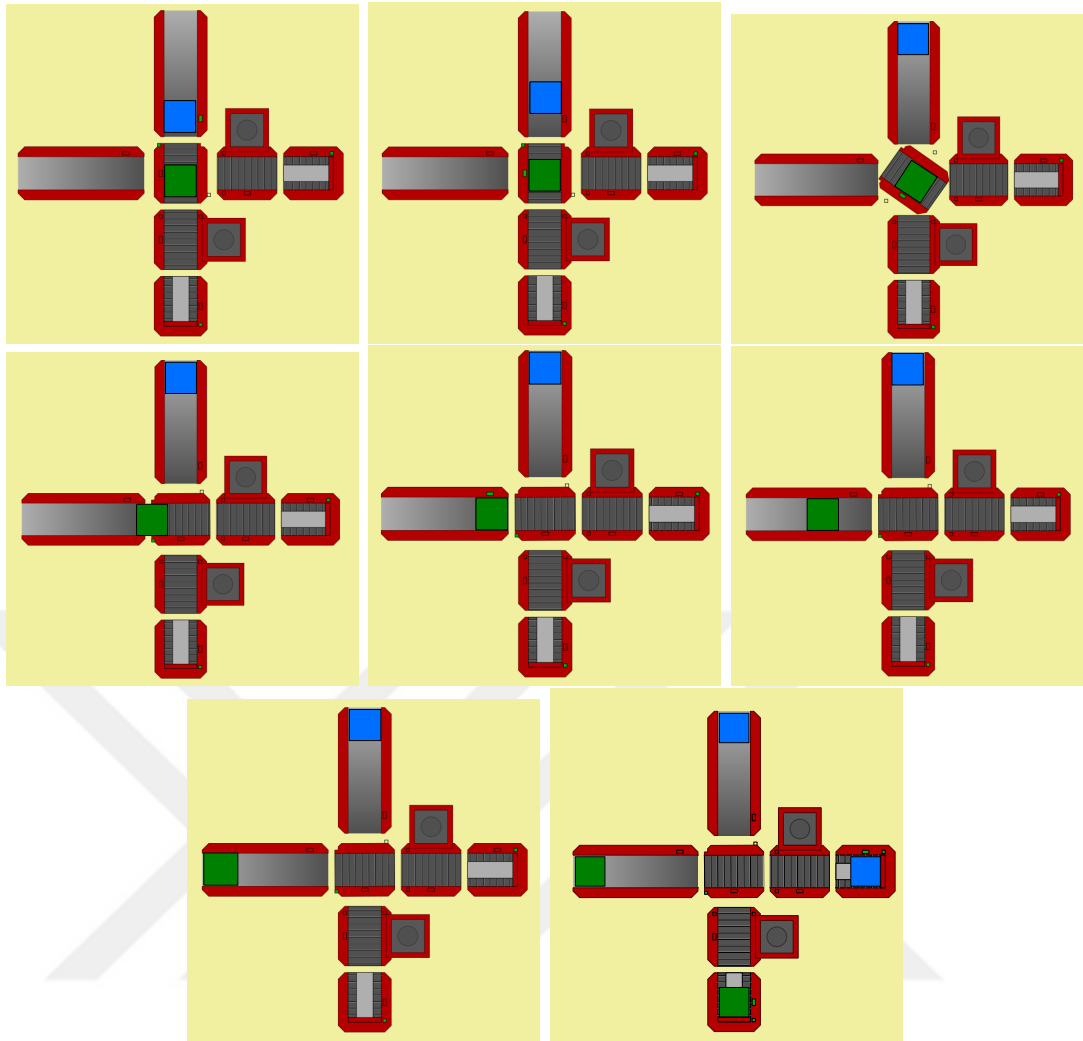


Figure 6.13: Snapshots of the FMS operation (continued).

6.2.2 FMS with a Long Conveyor Belt

We finally consider the FMS in Fig. 6.14 which is a modification of the small FMS in the previous section. It contains an additional long conveyor belt CB with a capacity of two products and a rotary table RT1.

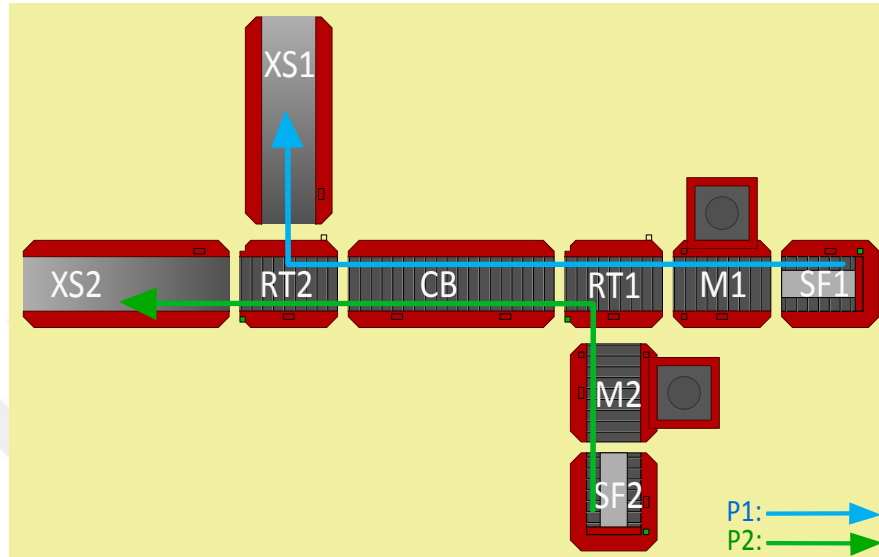


Figure 6.14: FMS model with different production components.

The models of M1, M2, SF1 and SF2 are identical to the models for the small FMS in Fig. 6.9. RT1 is now connected to the conveyor belt CB such that its model is modified to the automaton G_{RT1} in Fig. 6.15.

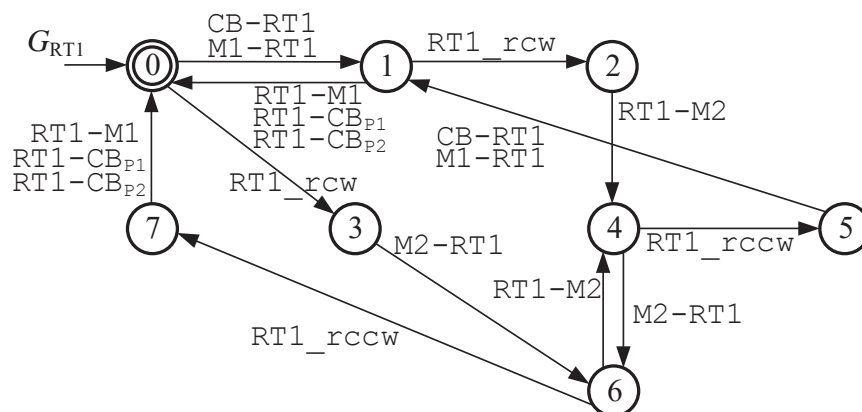


Figure 6.15: Model of RT1.

A similar model is used for the rotary table RT2 in Fig. 6.16.

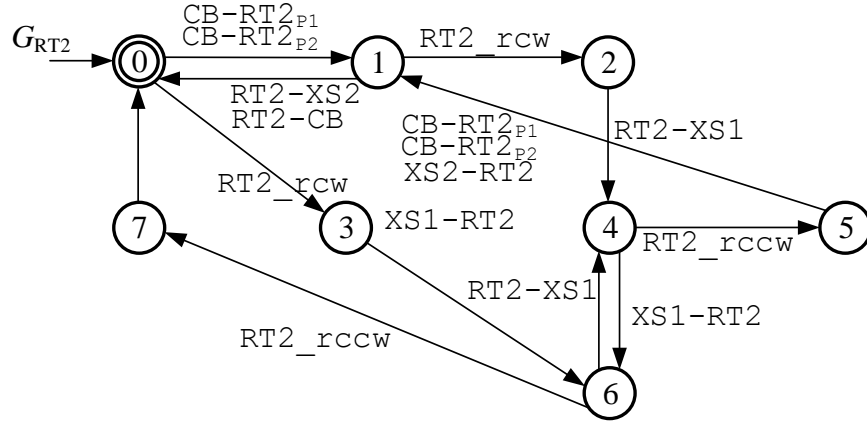


Figure 6.16: Model of RT2.

Finally, Fig. 6.17 shows the automata models of the long conveyor belt CB and the exit slides. Since CB has a capacity of 2, the model has 7 states in order to remember the order of incoming products of type P1 or P2.

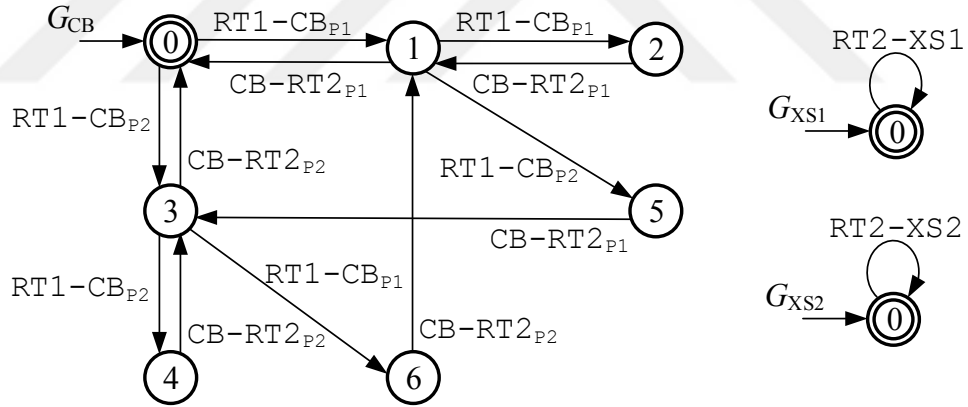


Figure 6.17: CB and exit slide models.

Using the previous models, the overall plant model of the extended FMS is given by

$$G_{FMS} = G_{SF1} || G_{SF2} || G_{M1} || G_{M2} || G_{RT1} || G_{CB} || G_{RT2} || G_{XS1} || G_{XS2}. \quad (6.5)$$

However, instead of computing a supervisor for the overall model, it is convenient

to divide the system into two modular components. We use the plant model

$$G_1 = G_{SF1} || G_{SF2} || G_{M1} || G_{M2} || G_{RT1} \quad (6.6)$$

to represent the modular component of the FMS until RT1. For this modular component, we employ the order-preserving specification C_1 in Fig. 6.18 similar to the small FMS.

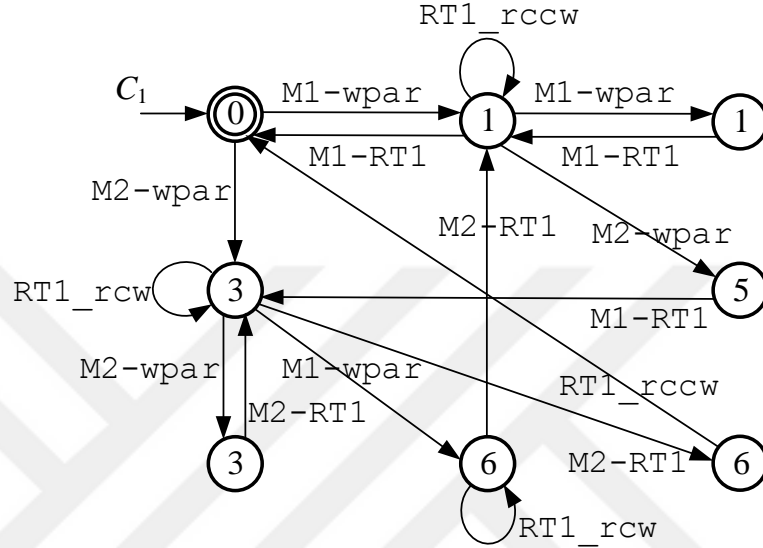


Figure 6.18: Order-preserving specification for the small FMS.

No additional specification is required for the second module

$$G_2 = G_{CB} || G_{RT2} || G_{XS1} || G_{XS2} \quad (6.7)$$

of the FMS since the order of products is already captured by the plant models. We finally compute modular supervisors S_1 and S_2 for the two modular components. For this example, it turns out that S_1 has 35 states and S_2 has 18 states. It is further ensured that the Overall closed loop is nonconflicting. Fig. 6.19 and 6.20 show snapshots of the FMS operation, preserving the order of products entering the system from SF1 and SF2.

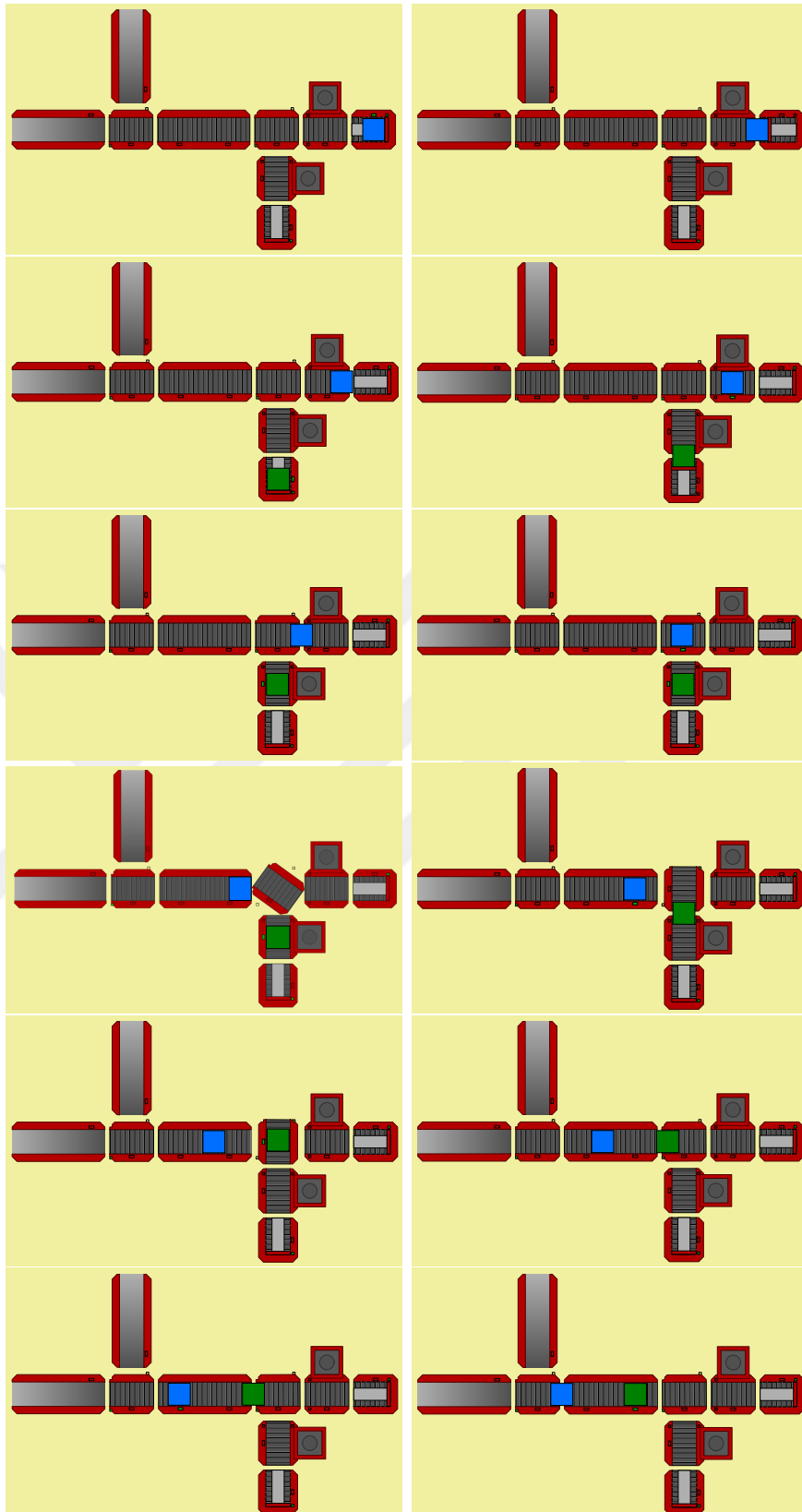


Figure 6.19: Snapshots of the FMS operation.

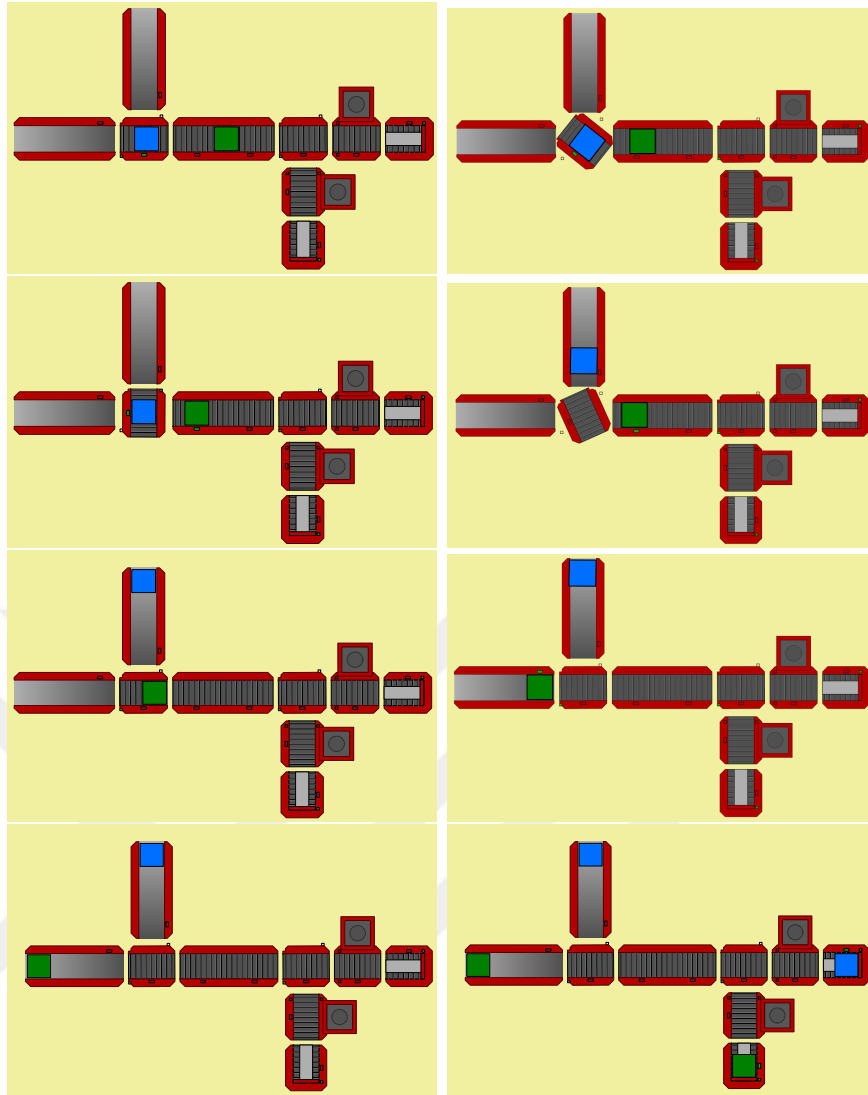


Figure 6.20: Snapshots of the FMS operation (continued).

CHAPTER 7

CONCLUSIONS

The subject of this thesis is the development of a new discrete event system (DES) model for the supervisory control of flexible manufacturing systems (FMS). The new model considers that FMS consist of various production components such as machines and robots that exchange products among each other. In addition, the model accounts for the fact that FMS are able to manufacture different product types that potentially share various production components such as machines and robots. Different from the existing literature, the proposed model also addresses the case where a production component can hold multiple products and then processes these products sequentially. Specifically, if the capacity of a production component is greater than one, any product that enters the production component first is processed first and also leaves the production component first.

In order to formalize this scenario, the thesis introduces the new class of order-preserving formal languages. We show that a supremal order-preserving language exists and can be represented by a finite state automaton. In addition, we prove that it is possible to compose order-preserving languages to again obtain an order-preserving language. Accordingly, we suggest to employ order-preserving language as a model for FMS that keeps track of the different product types entering and leaving production components. After presenting an algorithmic procedure for constructing such order-preserving model depending on the processed product types and the capacity of a production component, the thesis demonstrates the practicability of the proposed method by several FMS application examples, in-

cluding a comparison to existing models that allow products to enter and leave a production component in an arbitrary order.

Possible ideas for future work include the study of nonblocking system behavior depending on the connectivity of production components of order-preserving FMS.



REFERENCES

- [1] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.
- [2] D. N. Godbole, J. Lygeros, E. Singh, A. Deshpande, and A. E. Lindsey, “Communication protocols for a fault-tolerant automated highway system,” *IEEE Transactions on Control Systems Technology*, vol. 8, no. 5, pp. 787–800, Sep. 2000.
- [3] A. Benveniste, E. Fabre, S. Haar, and C. Jard, “Diagnosis of asynchronous discrete-event systems: a net unfolding approach,” *IEEE Transactions on Automatic Control*, vol. 48, no. 5, pp. 714–727, May 2003.
- [4] A. Bouloutas, G. W. Hart, and M. Schwartz, “Simple finite-state fault detectors for communication networks,” *IEEE Transactions on Communications*, vol. 40, no. 3, pp. 477–479, March 1992.
- [5] S. R. Das and L. E. Holloway, “Characterizing a confidence space for discrete event timings for fault monitoring using discrete sensing and actuation signals,” *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 30, no. 1, pp. 52–66, Jan 2000.
- [6] Shengbing Jiang, Zhongdong Huang, V. Chandra, and R. Kumar, “A polynomial algorithm for testing diagnosability of discrete-event systems,” *IEEE Transactions on Automatic Control*, vol. 46, no. 8, pp. 1318–1321, Aug 2001.
- [7] 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.
- [8] 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*. IGI Global, 2013, pp. 1–19.
- [9] 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.
- [10] 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.

- [11] 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.
- [12] 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.
- [13] 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.
- [14] 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.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] 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*. IEEE, 1994, pp. 2551–2556.
- [22] 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.
- [23] K. W. Schmidt, “Reconfigurability of behavioural specifications for manufacturing systems,” *International Journal of Control*, vol. 90, no. 12, pp. 2605–2617, 2017.

- [24] W. Wonham, *Supervisory Control of Discrete Event Systems*. Systems Control Group, University of Toronto, Canada, 2010.
- [25] 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.
- [26] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. Van Brussel, “Reconfigurable manufacturing systems,” *Annals of the CIRP*, vol. 48, p. 2, 1999.
- [27] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren, “Reconfigurable manufacturing systems: Key to future manufacturing,” *Journal of Intelligent manufacturing*, vol. 11, no. 4, pp. 403–419, 2000.
- [28] 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.
- [29] —, “On order-preserving languages and their application to flexible manufacturing systems,” *IEEE Access (in preparation)*, 2020.
- [30] 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.
- [31] T. Moor, K. Schmidt, and S. Perk, “libFAUDES - an open source c++ library for discrete event systems,” *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, pp. 125–130, May 2008.
- [32] libFAUDES. (2006–2010) libFAUDES software library for discrete event systems. [Online]. Available: www.rt.eei.uni-erlangen.de/FGdes/faudes

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Nooruldeen, Anas

Nationality: Iraqi (IRQ)

Date and Place of Birth: 20 June 1988 , Kirkuk

Marital Status: Single

Phone: +90 531 498 34 37/ +964 770 219 23 77

email: anasnooruldeen@gmail.com

EDUCATION

Degree	Institution	Year of Graduation
PhD	Çankaya Univ. Electronic and Communication Engineering	2020
MS	Çankaya Univ. Electronic and Communication Engineering	2012
BS	College of Technology/Kirkuk Electronic and Control Engineering	2010
High School	Kirkuk Centralized	2006

FOREIGN LANGUAGES

Arabic, English, Turkish

AWARDS & ACHIEVEMENTS

TÜBİTAK M.SC. Grant, 2011-2012.

TÜBİTAK Ph.D. Scholarship, 2016-2020.

Funding: THE SCIENTIFIC AND TECHNOLOGICAL RESEARCH COUNCIL OF TURKEY (TÜBİTAK).

PUBLICATIONS

A. Nooruldeen, K.W. Schmidt, State Attraction Under Language Specification for the Reconfiguration of Discrete Event Systems, *IEEE Transactions on Automatic Control*, 60(6), (2015), 1630–1634.

A. Nooruldeen and K.W. Schmidt, Order-preserving models for the supervisory control of flexible manufacturing systems, *Çankaya University Journal of Science and Engineering*, 16(2), (2019), 70–86.

A. Nooruldeen and K.W.Schmidt, On order-preserving languages and their application to flexible manufacturing systems, *IEEE Access* (in preparation), 2020.

