ÇANKAYA UNIVERSITY
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
ELECTRONIC AND COMMUNICATION ENGINEERING

MASTER THESIS

APPLICATIONS OF RECONFIGURABLE MANUFACTURING SYSTEMS: A LABORATORY CASE STUDY

HARITH M. KHALID HENDI

FEBRUARY 2014

Title of the Thesis: **Applications of Reconfigurable Manufacturing Systems: A Laboratory Case Study**

Submitted by **Harith M. Khalid Hendi**

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

Prof. Dr. Taner Altunok
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science

Prof. Dr. Celal Zaim ÇİL
Head of Department

This is to certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Klaus Werner SCHMIDT
Supervisor

Examination Date: 05 FEB 2014

Examining Committee Members

Dr. Zeki KOCABIYIKOĞLU	(Çankaya Univ.)	
Assoc. Prof. Dr. Klaus Werner SCHMIDT	(Çankaya Univ.)	
Buğra Sağlam	(TAI)	

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 : Harith M. Khalid Hendi

Signature :

Date : 05 FEB 2014

ABSTRACT

APPLICATIONS OF RECONFIGURABLE MANUFACTURING SYSTEMS: A LABORATORY CASE STUDY

HENDI, Harith M. Khalid

M.Sc., Department of Electronics and Communication Engineering

Supervisor: Assoc. Prof. Dr. Klaus Werner SCHMIDT

February 2014, 122 pages

Reconfiguration control for discrete events system (DES) is applicable to the design of industrial manufacturing systems in order to support changes in the variety and the quantity of products. Hereby, reconfiguration control is concerned with the realization of different system configurations that become active on request. A manufacturing system that is subject to reconfiguration control is denoted as a reconfigurable manufacturing system (RMS). In practice, RMS are of large size with many manufacturing components and hence come with a large design complexity.

In this thesis, we develop a method for the reconfiguration control of large-scale RMS and reconfigurable machine tools (RMT). We propose to first construct modular reconfiguration supervisors that realize the desired configurations and configuration changes for modular components of the overall RMS. Then, we apply abstraction-based control in order to combine the operation of the different RMS modules. As a result, we obtain a hierarchy of reconfiguration supervisors that allow changes to any desired configuration at any time. Since, we use the technique of hierarchical abstraction, our method is applicable to large-scale RMS, different from all existing approaches in the literature that focus on a monolithic design. We demonstrate the applicability of our method by a large-scale laboratory RMS with 18 manufacturing components and a controller hierarchy with 4 levels.

Keywords: Reconfigurable manufacturing systems, Reconfigurable machine tools, Discrete events systems, Supervisory control, Abstraction.

ÖZ

Yeniden Yapılandırılabilir Üretim Sistemleri Uygulamaları: Bir Örnek Laboratuvar Calışması

HENDI, Harith M. Khalid

Yüksek Lisans, Elektronik ve Haberleşme Mühendisliği

Tez Yöneticisi: Assoc. Prof. Dr. Klaus Werner SCHMIDT

Şubat 2014, 122 pages

Ayrık olaylı sistemlerde yeniden yapılandırılabilir kontrol, üretim sistemlerinin ürün çeşitliliği ve kalitesini değiştirmek için kullanılır. Bu noktada, yeniden yapılandırılabilir kontrol, istek yapıldıktan sonra farklı sistem konfigürasyonlarının gerçekleştirilmesini amaçlamaktadır. Yeniden yapılandırılabilir kontrol yapısını kullanan üretim sistemleri, yeniden yapılandırılabilir üretim sistemleri (reconfigurable manufacturing systems - RMS) olarak adlandırılmaktadır. Yeniden yapılandırılabilir üretim sistemleri, farklı birçok üretim birimlerinden oluşabilmekte ve bunun sonucunda yüksek tasarım karmaşıklığına sahip olabilmektedir.

Bu tezde, genişölçekli yeniden yapılandırılabilir üretim sistemleri tasarımı ve yeniden yapılandırılabilir makina araçları (reconfigurable machine tool - RMT) tasarımı için bir metod geliştirilmiştir. Öncelikle, istenen konfigürasyonlar için modüler yeniden yapılandırılabilir kontrolcüler oluşturulmuştur. Sonrasında, farklı konfigürasyonları birleştirmek için model boyutlarını küçülten soyutlama tabanlı kontrol algoritması kullanılmıştır. Sonuçta, herhangi bir zamanda istenilen konfigürasyona geçişi sağlayan hiyerarşik yapıda yeniden yapılandırılabilir kontrolcü elde edilmiştir. Hiyerarşik yeniden boyutlama metodu kullanıldığı için, literatürde

tanımlı monolitik çözümlerin aksine farklı birçok birimden oluşan geniş ölçekli yeniden yapılandırılabilir üretim sistemlerine uygulanabilen bir metod geliştirilmiştir. Bu metodun uygulanabilirliğini göstermek için, laboratuvarda, 18 farklı birimden ve 4 seviye hiyerarşik kontrolcüden oluşan geniş ölçekli yeniden yapılandırılabilir üretim sistemi modellenmiştir.

Anahtar Kelimeler: Yeniden yapılandırılabilir üretim sistemleri, Yeniden yapılandırılabilir makina araçları, Ayrık olaylı sistemler, Kontrolcü taşarımı, Yeniden boyutlama.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my thesis advisor Assoc.Prof. Dr. Klaus Werner SCHMIDT, who has encouraged and guided me throughout this thesis patiently.

I also would like to express my deepest gratitude to the force, my father whose support makes my way of successes. To the big heart, my mother whose prayers makes me hopeful in my life.

I also would like to express my sincere gratitude to my aunt, who supported and encouraged me along my life. To the happiness of my life, my lovely sisters and brothers.

I also thank and appreciate the Scientific and Technological Research Council of Turkey (TÜBİTAK), in the provision of material and financial support (during my thesis). Note, that this thesis was supported by TÜBİTAK [Career Award 110E185].

I also thank the University of Çankaya, and especially the department of mechatronics engineering , as well as the department of electronic and communication engineering, for their support and create the appropriate requirements of laboratories and equipment necessary during periods of study and research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DES	Discrete events systems
DML	Dedicated manufacturing lines
FMS	Flexible manufacturing systems
RMS	Reconfigurable manufacturing systems
RMT	Reconfigurable machine tools

INTRODUCTION

Traditional manufacturing systems can be classified into *dedicated manufacturing lines* (DML) and *flexible manufacturing systems* (FMS), whereby both types produce products with a fixed system structure. In contrast, the new type of *reconfigurable manufacturing systems* (RMS) is developed in order to keep pace with the evolution in the field of manufacturing industries. Reconfiguration control gives the ability to change the system configuration depending on the product demand in terms of product type and quantity [11, 15, 4, 10]. Application areas for RMSs can for example be determined in the production of automobile parts [8, 4, 3].

The controller design for RMS requires several properties listed as follows.

- In each individual system configuration, the desired operation should be realized.
- If a configuration change is requested, then the change has to be possible and it should be realized as fast as possible.
- A suitable controller design method must be usable for large-scale systems.

In the existing literature, there are different methods for the reconfiguration control of DES. The realization of reconfiguration supervisors was studied without a synthesis algorithm in [5]. Later, synthesis approaches for reconfiguration controllers were realized in [12, 13, 18, 7, 6] according to different assumptions, whereby all the cited methods only focus on monolithic synthesis. That is, none of the existing methods is applicable to large-scale RMS that are typically encountered in practice.

A new line of research on RMS in the framework of supervisory control for discrete event systems (DES) was initiated in the scope of the TÜBİTAK project 110E185 "A Formal Framework and Continuous Workflow for the Controller Design, Failure Diagnosis and Failure Recovery of Reconfigurable Manufacturing Systems". In the scope of this project, also several approaches for the monolithic controller design

for RMS are developed [16, 21, 22, 24, 23]. The work in this thesis extends these approaches to a design method that is suitable for large-scale RMS.

The main subject of this thesis is a new method for the design of reconfiguration supervisors for large-scale RMS that comprise multiple components. Such RMS is divided into smaller modules and the idea of abstraction-based supervisory control [20, 19] is applied to obtain modular supervisors that realize each system configuration. In addition, a modified version of state attraction supervisors [1, 2] is employed in order to achieve completion of an existing configuration whenever a new configuration is requested. Combining the abstraction-based supervisors and the state attraction supervisors, the thesis presents an algorithm for the computation of a modular reconfiguration supervisor for each RMS module. These modular reconfiguration supervisors have a small size, which makes the proposed method applicable to large-scale RMS. In addition, the thesis develops a modeling framework for reconfigurable machine tools (RMT) that are basic building blocks of RMS.

In summary, the main contributions of the thesis are

- the application of abstraction-based supervisory control to RMS.
- the generalization of state attraction to *modular state-attraction* that can be applied without a monolithic system model.
- a construction algorithm for modular reconfiguration supervisors.
- a modeling framework for RMTs.

In addition, the developed method is applied to a large-scale laboratory model of an RMS with 18 components. The resulting design leads to a controller hierarchy with 4 levels and shows considerable computational savings compared to a monolithic design. Here, we do not only compute the supervisors for this RMS example but we also evaluate the correctness of the design by extensive simulations of the RMS behavior. This simulation underlines that the realization of our reconfiguration supervisors never needs the evaluation of a monolithic model of the system. Some of the results in the thesis are published in the conference paper [9] that also formally states the correctness of the proposed design.

The remainder of this thesis is organized such that Chapter I contains all the basic notation about discrete event systems, supervisory control, abstraction-based con-

trol. Chapter II gives all discussion about the modeling and abstraction of RMT. It contains the modeling of the RMT in Section 2.1, and the reconfiguration control for the RMT in Section 2.2. Moreover, the abstraction-based reconfiguration controller design for RMS is developed in Chapter III. Here, Section 3.2 considers the design for a single RMS module and Section 3.3 introduces hierarchical abstraction. Chapter IV presents the details of a large application example including an RMS simulation.

CHAPTER I

BASIC NOTATION

In this chapter, we present the basic notation and the background literature. The chapter is organized as follows, Section 1.1 shows the concept of discrete events system DES, Section 1.2 gives the basic notation in regard to the formal language. In Section 1.3 automata are introduced and Section 1.4 describes the supervisory control of discrete events system DES. In Section 1.5 the concept of state attraction, strong attraction is outlined, while section 1.6 explains the approach of abstraction-based supervisory control and decentralized control. Finally, all definitions about the natural observer projections are given also in Section 1.6.

1.1 DISCRETE EVENTS SYSTEMS

The concept of discrete event systems (DES) is introduced for systems whose state space is a discrete Set, such as {0,1,2,3,...}. For such systems, state transitions are possible at discrete times and they are triggered by the occurrence of discrete events.

Characteristic properties of discrete event systems are

- Finite number of *discrete states* where the system spends time.
- The *discrete events* occur instantaneously.
- The *discrete transitions* change the system state due to occurrence of events.

DES are widely used in different domains and have been applied in various fields. Such examples of discrete events systems are manufacturing systems, traffic systems, and computer systems. A lot of system's performance and behaviors can be clarified by DES model, for example an operation of a device (ON,OFF), a

message to travel from source to destination in communication systems or calls as in telephone networks.

A small example of DES is a simple machine with

- Three discrete states as modes of operation {IDLE, BUSY, DOWN}.
- Four discrete events {START, BREAK, REPAIR, FINISH}.
- Four discrete transitions.

Initially, the machine is in the first state *IDLE* and the possible transition *START* leads to the second state *BUSY*. From this state there are two possible transitions. First, *FINISH* leads back to the IDLE state. Second, *BREAK* leads to the state *DOWN*, from where a transition *REPAIR* leads to the IDLE state. That means, the machine starts when it is idle and finishes after some time. The machine can break during operation and be repaired after breakdown, finally the machine restarts after the repair happened.

A formal language can be used to model and shape the behavior of DES as described in the next section.

1.2 FORMAL LANGUAGE

We consider the concept of *languages* to model the behavior of DES. The finite set of events is called *alphabet* and is denoted as Σ. A finite event sequence from Σ is called a *string*. For each string s, $|s|$ is written the length of this string such that $|s|$ defines the number of events in s. ϵ is the empty string with length $|\epsilon| = 0$, such that $s\epsilon = \epsilon s = s$. The *kleene closure* Σ^* is the set of all finite strings over a finite alphabet Σ.

Now we recall the example of a *simple machine*. Here, the alphabet is Σ ={start, break, repair, finish}. Example strings are given by s_1 =start finish with length $|s_1| = 2$ or s_2 =start break repair start finish with length $|s_2| = 5$. The Kleene closure for this example is written as $\Sigma^* = \{\epsilon$, start, break, repair, finish, start finish, start break repair, start break repair start finish, ...,$\}$.

The *concatenation* of two strings $s_1, s_2 \in \Sigma^*$ is written as $s = s_1 s_2$ and s_1 is a prefix of s and s_2 is a suffix of s. A language over Σ is a subset $L \subseteq \Sigma^*$. The

prefix closure $\overline{L}$ of L is defined as $\overline{L} = \{s_1 \in \Sigma^* | \exists s \in L \text{ s.t. } s_1 \leq s\}$. If $L = \overline{L}$, then the language $L \in \Sigma^*$ is called prefix closed. For string $s \in L$ and a language $L \in \Sigma^*$ the set of suffixes of s in L is written as $L/s = \{t \in \Sigma^* | \exists st \in L\}$.

For the *natural projection* we first introduce $\hat{\Sigma}$ as a subset of Σ. Then, the natural projection removes all events in Σ that do not belong to $\hat{\Sigma}$. The natural projection is written as $p : \Sigma^* \to \hat{\Sigma}^*$ such that

$p(\epsilon) = \epsilon$; $p(\sigma) = \sigma$ if $\sigma \in \hat{\Sigma}$ and $p(\sigma) = \epsilon$ otherwise; $p(s\sigma) = p(s)p(\sigma)$ for $s \in \Sigma^*$ and $\sigma \in \Sigma$.

The inverse projection is given as $p^{-1} : \Sigma^* \to 2^{\Sigma^*}$ such that for each $t \in \Sigma^*$: $p^{-1}(t) = \{s \in \Sigma^* | p(s) = t\}$.

Consider that $\hat{\Sigma}$={start} for the simple machine example. If the natural projection is applied to the string s_2 = start break repair start finish, then the result is $p(s_2)$ = start start. All other events are removed by the projection.

1.3 AUTOMATA

DES can be modeled by a finite state automaton G that is represented by a five-tuple $G = (X, \Sigma, \delta, x_0, X_\mathrm{m})$ with the finite set of states X; a finite set of events Σ; a partial transition function $\delta : X \times \Sigma \to X$; the initial state denoted by $x_0 \in \Sigma$ and the marked states $X_\mathrm{m} \subseteq X$. Here, the marked states are desired states that should be reached.

Automata can be graphically represented by state transition diagrams or graph representations. The automata model for the simple machine example is given by the simple state transition diagram shown in fig. 1.1. Here circles represent states, arrows represent transitions that are labeled by events, the incoming arrow indicates the initial state and double circles represent marked states.

Two subset languages of Σ^*, the *closed language* $L(G)$ and the *marked language* $L_m(G)$ are used to characterized the finite states automaton G. All event sequences that follow transitions starting from the initial state are included in $L(G)$ and all stings starting from the initial state and leading to a marked state will be in $L_m(G)$. If $\overline{L_m(G)} = L(G)$, the automaton G is called *nonblocking*. This means that every string in G can be extended to a marked state. Closed language and

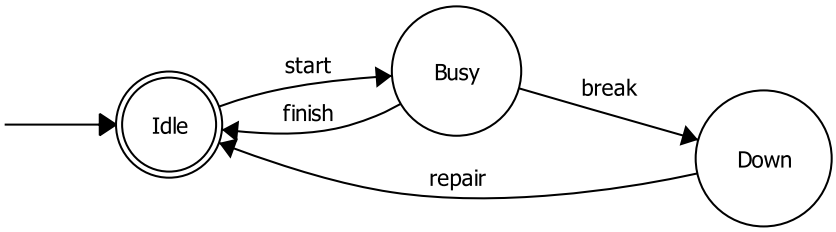

Figure 1.1: Simple machine

marked language are formally defined by

$$L(G) = \{s \in \Sigma^* | \exists \delta(x_0, s)!\} \tag{1.1}$$

$$L_m(G) = \{s \in L(G) | \exists \delta(x_0, s) \in X_m\} \tag{1.2}$$

In a finite state automaton G, if a string starts from a state and follows transitions in G back to the same state, then G has a cycle. If an auotmaton G has no cycles, it is called acyclic.

Important automata operations are :

- Accessible: G is called accessible if all states in X can be reached from x_0:

  $$\forall x \in X, \exists s \in \Sigma^* \text{ s.t. } \delta(x_0, s) = x \tag{1.3}$$

  The operation Acc(G) is used to make an automaton G accessible by removing all states that are not reachable from the initial state x_0.

- Coaccessible: G is called coaccessible if a marked state in X_m can be reached from any state of X:

  $$\forall x \in X, \exists s \in \Sigma^* \text{ s.t. } \delta(x_0, s) \in X_m \tag{1.4}$$

  The operation CoAcc(G) is used to make an automaton G coaccessible by removing all states in G from where no marked state is reachable. It has to be noted that all coaccessible automata are nonblocking.

- Trim: If an automaton G is accessible and coaccessible at the same time, then G is trim. Such that Trim(G) = ACC(CoAcc(G)) = CoAcc(Acc(G)).

Our simple machine example automaton G=(X, Σ, δ, x_0, X_m) has three states, four events in it's alphabet, and four transitions such that X = {Idle, Busy,

Down}; Σ ={start, break, repair, finish}; δ is defined by δ (Idle, start) = Busy; δ (Busy, finish) = Idle; δ (Busy, break) = Down; δ (Down, repair) = Idle. Furthermore, x_0 = Idle is the initial state and X_m ={Idle} as the set of marked states.

Here, all states are reachable from x_0 and X_m is reachable from any state. This means that an automata G is accessible and coaccessible and hence also trim. It can also be seen that G is cyclic.

Consider automata $G = (X, \Sigma, \delta, x_0, X_{\mathrm{m}})$ and $G' = (X', \Sigma, \delta', x_0', X_{\mathrm{m}}')$. G' is a subautomaton of G written as $G' \sqsubseteq G$, if G' is an empty automaton ($X' = 0$) or $X' \subseteq X$, $x_0' = x_0$ and for all $x \in X'$ and $\sigma \in \Sigma$, holding that $\delta'(x, \delta)! \Rightarrow \delta'(x, \delta) = \delta(x, \sigma)$. This means that G' is obtained by removing states and transitions from G.

Finally, we consider discrete event systems with multiple components (more than one component). Such systems can be modeled by multiple finite state automata. Then, the overall model as one single automaton is obtained by applying the synchronous composition, This composition synchronizes the occurrence of events that are shared in at least two automata, while other events happen independently. let $G_1 = (X_1, \Sigma_1, \delta_1, x_{0,1}, X_{\mathrm{m},1}$ and $G_2 = (X_2, \Sigma_2, \delta_2, x_{0,2}, X_{\mathrm{m},2})$ be two automata. The synchronous composition is described as:

$$G_1||G_2 = G_{12} = (X_{12}, \Sigma_{12}, \delta_{12}, x_{0,12}, X_{\mathrm{m},12}) \tag{1.5}$$

The synchronous composition defined such that the states are ($X_{12} = X_1 \times X_2$; events are $\Sigma_{12} = \Sigma_1 \cup \Sigma_2$; initial state is $x_{0,12} = (x_{0,1}, x_{0,2})$; marked states are $X_{\mathrm{m},12} = X_{\mathrm{m},1} \times X_{\mathrm{m},2}$; and the transition relation is defined for $(x_1, x_2) \in X_{12}$ and $\sigma \in \Sigma_{12}$ as

$$\delta_{12}((x_1, x_2), \sigma) = \begin{cases} (\delta_1(x_1, \sigma), \delta_2(x_2, \sigma) & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2 \wedge \delta_1(x_1, \sigma)! \wedge \delta_2(x_2, \sigma)! \\ (\delta_1(x_1, \sigma), x_2) & \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \wedge \delta_1(x_1, \sigma)! \\ (x_1, \delta_2(x_2, \sigma)) & \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \wedge \delta_2(x_2, \sigma)! \end{cases} \tag{1.6}$$

1.4 SUPERVISORY CONTROL

In [17] Ramadge and Wonham introduce the theory of supervisory control for DES. The concept of this theory is how to design and apply control for DES. The

controller is denoted as supervisor and is capable of allowing or preventing the occurrence of some events in the DES in order to achieve the desirable behavior.

Consider G=(X, Σ, δ, x_0, X_m) a deterministic finite state automaton as a DES plant, Σ_c is a controllable events set, Σ_u is an uncontrollable events set. Then the alphabet of the plant Σ can be described as

$$\Sigma = \Sigma_c \cup \Sigma_u \tag{1.7}$$

The supervisor S is also realized by a finite state automaton as

$$S = (Q, \Sigma, \nu, q_0, Q_{\mathrm{m}}) \tag{1.8}$$

This supervisor S, can disable all events in Σ_c the controllable events, but for the uncontrollable events in Σ_u can not be disabled. In the simple machine example, the "start" event is an example for a controllable events, whereas the "break" event is an uncontrollable events.

Here, the synchronous composition $G||S$ of the plant G and the supervisor S will give the desired behavior of the controlled system (closed loop behavior). Furthermore $L(G)||L(S)$ will be the closed language for the closed loop, and $L_{\mathrm{m}}(G)||L_{\mathrm{m}}(S)$ the marked language of the closed loop. If $G||S$ is nonblocking then S will be nonblocking supervisor. Also, for all $s \in L(G) \cap L(S)$ and $\sigma \in \Sigma_{\mathrm{u}}$ with $s\sigma \in L(G)$ and we have $s\sigma \in L(S)$. This is because S is not allowed to disable events in Σ_u, and if σ can happen after s in G, then σ must be possible after s in S.

Until now, we have G as a DES plant and S as a supervisor. It is possible to specify the desired closed-loop behavior using an automaton $C = (Y, \Sigma, \beta, y_0, Y_{\mathrm{m}})$ whose language $K = L_{\mathrm{m}}(C)$ is denoted as the specification language. The specification contains all desirable strings such that the supervisor must disable all undesired strings that are in the plant but not in the specification. This is only possible if the supervisor does not need to disable uncontrollable events. That is, the specification K should be controllable w.r.t. plant G and uncontrollable events Σ_u. Formally, this holds if

$$\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K} \tag{1.9}$$

Here, $\overline{K}$ is the prefix closure of K, and $\overline{K}\Sigma_u$ describes the set of strings, that start with a prefix in $\overline{K}$ and are followed by an event from Σ_u. In this case, any string

belonging to the specification K and that can be extended by an uncontrollable event and that belongs to the plant G should also belong to the specification.

If the specification K is not controllable with respect to plant G and uncontrollable events Σ_u, then the *supremal controllable sublanguage* of K should by implemented. The concept of this *SupCon* algorithm is to find $K_{sub} \subseteq K$ such that K_{sub} is controllable for G and Σ_u, where K_{sub} is the largest possible sublanguage, and $SupC(K, L(G), \Sigma_u)$ contains all controllable sublanguages of K.

$$SupC(K, L(G), \Sigma_u) = \bigcup \{K' \subseteq K | K' \text{ controllable for } G \text{ and } \Sigma_u\} \tag{1.10}$$

Also, we can write $L_m(S//G) = SupC(K, L(G), \Sigma_u)$ and we know that such supervisor is nonblocking.

1.5 STATE ATTRACTION

In order to discuss the state attraction concept, we will introduce *strong attraction* and *weak attraction* notions. Let $G = (X, \Sigma, \delta, x_0, X_m)$ be a finite state automaton representing the plant of DES, Σ_u as a set of uncontrollable events, *invariant set* and a *weakly invariant set* $X' \subseteq X$.

$X' \subseteq X$ is an invariant set in G if no outgoing transition from the states of X' leads to states that do not belong to X' described as, $\forall x \in X'$ and $\sigma \in \Sigma$ holds that $\delta(x, \sigma)! \Rightarrow \delta(x, \sigma) \in X'$.

We next define the concept of strong and weak attraction.

Definition 1. *Let $A \subseteq X' \subseteq X$ given that A, X' are invariant sets in G. A is a* strong attractor *for X' in G if*

- *the (strict subautomaton) of G with state set $X' \setminus A$ is acyclic*

- *$\forall x \in X'$, there is $u \in \Sigma^\star$ s.t. $\delta(x, u) \in A$*

That means that A is reached from any state in X' after the occurrence of a bounded number of events.

Definition 2. *Let $A \subseteq X' \subseteq X$ and assume A, X' are invariant sets in G, uncontrollable set Σ_u. A is described as a* weak attractor *for $X' \in G$ if there exists a state-feedback supervisor $S \sqsubseteq G$ s.t. for X' in S, A is a strong attractor.*

Next, let $\Omega_G(A) \subseteq X$ be the *supremal subset* of X such that A is a weak attractor for $\Omega_G(A)$ in the plant G with uncontrollable events set Σ_u. The *supremal subset* algorithm can be used to compute $\Omega_G(A)$. The complexity for this algorithm is $\mathcal{O}(|X| \cdot |\Sigma|)$ and as we know $|X|$ described states number and $|\Sigma|$ described the events number. Notice that this algorithm with complexity $\mathcal{O}(|X|^2)$ will be used to find S, when $S \sqsubseteq G$ and A is a strong attractor for $\Omega_G(A)$.

1.6 ABSTRACTION-BASED SUPERVISORY CONTROL

The combination of *decentralized control* and *hierarchical control* can be applied to DES with many components in order to handle large-scale DES. Such approach is important, considering that the number of plant states grows exponentially with the number of the system components (*state space explosion problem*). The following architecture is used to perform computations with small automata.

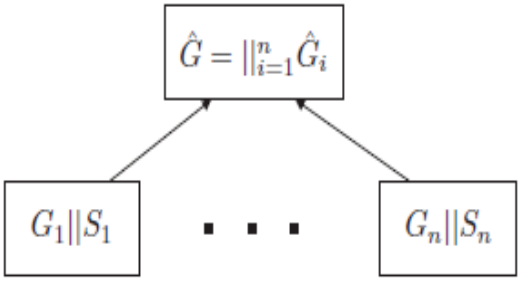

Figure 1.2: Hierarchical and decentralized control architecture.

Let $G_i, i = 1, ..., n$ be n finite plant automata for a DES with more than one component over the corresponding alphabets $\Sigma_i = \Sigma_{i,c} \cup \Sigma_{i,u}$, the controllable events denoted by $\Sigma_{i,c}$ and the uncontrollable events by $\Sigma_{i,u}$.

Implementation of abstraction required to introduce the abstraction alphabet also called *high-level alphabet*. Here, it is important to preserve the shared behavior in the abstracted level. Hence, we introduce the set of shared events of each component as $\Sigma_{i,\cap} = \bigcup_{k=1,k\neq i}^{n} (\Sigma_i \cap \Sigma_k)$.

For the overall system, $G = \|_{i=1}^{n} G_i$ is the overall plant model and $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$ is the overall alphabets, while the global shared events are given as $\Sigma_\cap = \bigcup_{i=1}^{n} \Sigma_i$. Moreover $\forall i, k, i \neq k, \Sigma_{i,u} \cap \Sigma_{k,c} = 0$ it must hold $\sigma_u = \bigcup_{i=1}^{n} \Sigma_{i,u}$ and $\Sigma_c = \bigcup_{i=1}^{n} \Sigma_{i,c}$. In words, the controllability status of these shared events of different components must be the same.

As we noticed before in previous section, the desired behavior for any DES can be achieved by computing a according to the given specification (K). In this case the desired behavior will given by multiple specifications $K_i \subseteq L_m(G_i), i = 1, ..., n$, and the overall specification will be $\hat{K} \subseteq \hat{\Sigma^*}$, by assuming that $\Sigma_\cap \subseteq \hat{\Sigma} \subseteq \Sigma$.

Using this setting, the abstraction approach proceeds as follow

- Computation of the components supervisors S_i, $i = 1, ..., n$, such that

$$L_m(S_i) = SupC(K_i, L(G_i), \Sigma_{u,i} \tag{1.11}$$

- Using the projections $p_i : \Sigma_i^* \to \Sigma_i \cap \hat{\Sigma^*}$ to get the abstraction for the obtained low-level closed loop $(G_n \ // \ S_n)$. Then $\hat{G_i}$, $i = 1, ..., n$, are the abstracted closed loops that serve as plants for the high level.

- The parallel composition is applied to obtain the overall abstracted plant $\hat{G} = \|_{i=1}^{n} \hat{G_i}$. Then $\hat{S}$ the abstraction-based supervisor is computed using the given specification $\hat{K}$, as

$$L_m(\hat{S}) = SupC(\hat{K}, L(\hat{G}), \Sigma_u \cap \hat{\Sigma}) \tag{1.12}$$

- The final result for the overall closed loop is given by

$$\hat{S}||(\|_{i=1}^{n} S_i||G_i) \tag{1.13}$$

The described procedure leads to nonblocking control if the projections p_i that are used for the abstraction have the natural observer property:

Definition 3. *Considering of G an automaton, Σ as the alphabet, and let that $\hat{\Sigma} \subseteq \Sigma$, then define $p : \Sigma^* \to \hat{\Sigma^*}$ as a natural projection. Here p is an* Natural observer *if $\forall s \in L(G)$ and $\forall t \in \hat{\Sigma^*}$ it holds that*

$$p(s)t \in p(L_m(G)) \Rightarrow \exists u \in \hat{\Sigma^*} \ s.t. \ p(u) = t \ and \ su \in L_m(G) \tag{1.14}$$

In words, for the projection of string s extended to a marked string in the marked language of $(\hat{G})$ by t, then the original strings also must have an extension u to a marked string in $L(G)$.

If all the projections p_i are natural observers, then the closed-loop system for the abstraction-based design is nonblocking.

Theorem 4. *Regarding the computation of abstraction-based supervisors as described above, if all projections natural observers, then the obtained supervisors are nonblocking:*

$$\overline{L_m(\hat{S}||(\|_{i=1}^{n} S_i||G_i))} = L(\hat{S}||(\|_{i=1}^{n} S_i||G_i)) \tag{1.15}$$

CHAPTER II

RECONFIGURABLE MANUFACTURING TOOL (RMT): MODELING AND ABSTRACTION

2.1 RMT MODELING

The *reconfigurable manufacturing tool* (RMT) is one of the main building blocks of a *reconfigurable Manufacturing system* (RMS). It has to be designed such that its structure change can be done rapidly. In this thesis, we focus on an RMT example from an RMS laboratory model as is shown in Figure 2.1.

Figure 2.1: Picture of the RMT example system.

The RMT is in the middle of two neighbor components, denoted as R for the right neighbor and L for the left neighbor. We write MA for the RMT that is composed of three main components:

- One *conveyor belt*: The main function of the conveyor belt is to move products to MA (product input) and remove products from MA (product output).

- One *machine head*: The machine head holds three different machine tools. It can rotate in order to make each desired machine tool ready for operation. In addition, the machine head can move up and down in order to move the active machine tool toward a product that is located on the conveyor belt.

- Three *machine tools*: Each machine tool is responsible for a different mechanical processing operation. In our system, we consider drilling (D), milling (M) and polishing (P).

We next develop an overall model of the RMT from models of its components. Hereby, we use the discrete event formalism as introduced in Chapter I.

2.1.1 The Conveyor Belt Model

The hardware component of the conveyor belt that can be shown in Figure 2.2, contains one actuator motor and one sensor that is positioned in the middle of the conveyor belt. The purpose of the motor is to move products to and from the conveyor belt, whereby motion to the right and to the left are possible. The sensor detects if a product arrives at or leaves from the conveyor belt. Hence, the conveyor belt manages the input and output of products to the RMT. In order to obtain an automata model of the conveyor belt, we introduce the relevant events in the Table 2.1.

Figure 2.2: Picture of the conveyor belt.

Using these events, the automaton G_{Belt} in Figure 2.3 represents the discrete event

Table 2.1: Conveyor belt events.

EVENTS NAMES	DESCRIPTION	STATUS
R-ma_SW	move products (right to RMT)	C
ma-R_SW	move products (RMT to right)	C
L-ma_SW	move products (left to RMT)	C
ma-L_SW	move products (RMT to left)	C
ma_bm+	motor switch on (right direction)	C
ma_bm-	motor switch on (left direction)	C
ma_wpar	sensor detects product arrives	unC
ma_boff	motor switch off (stop)	C
ma_wplv	sensor detects product leaves	unC
ma_lvtoR	leave to right neighbor	unC
ma_lvtoL	leave to left neighbor	unC

model of the conveyor belt. Hereby, state #1 represents the empty conveyor belt without motion and state #6 represents the conveyor belt holding a product.

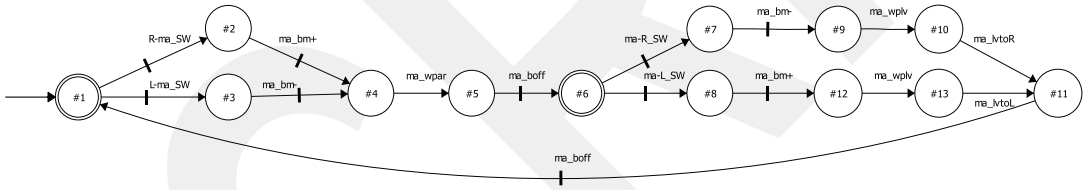

Figure 2.3: G_{Belt}

2.1.2 Machine Head Model

This hardware component contains two motors and three sensors. The machine head can move vertically from its upper rest position to the lower operational position and vice versa. This motion is supported by two sensors that are positioned in the upper and lower position, respectively. In addition, the machine head can turn to three diffident positions, each position according to the individual operation of the corresponding machine tool. The arrival of a tool at the center position is detected by a sensor. In summary the machine head provides a vertical and a rotational motion. We present automata models for both motions as shown in Figure 2.4 and 2.5. Hereby, $G_{vertical}$ in Figure 2.4 represents the vertical motion and $G_{rotation}$ in Figure 2.5 represents the rotational motion.

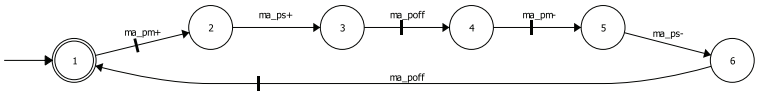

Figure 2.4: $G_{vertical}$

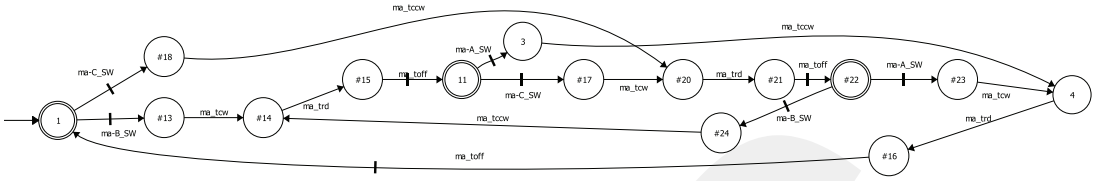

Figure 2.5: $G_{rotation}$

The related events are explained in Table 2.2.

Table 2.2: Machine head events

EVENTS NAMES	DESCRIPTION	STATUS
ma_pm+	motor on and moves (up-down)	C
ma_ps+	machine head arrives at upper position	unC
ma_poff	motor off (stop)	C
ma_pm-	motor on and moves (down-up)	C
ma_ps-	machine head arrives at lower position	unC
ma-A_SW	turns to position (operation A)	C
ma-B_SW	turns to position (operation B)	C
ma-C_SW	turns to position (operation C)	C
ma_tcw	turns the head (clockwise)	C
ma_tccw	turns the head (counter clockwise)	C
ma_toff	switch off (stop)	C
ma_trd	acknowledgment of finish	C

2.1.3 Machine Tool Model

Since the RMT comprises several machine tools, it can realize more than one operation. The operation of each tool requires the rotation of the respective tool that is performed by a motor. Since our RMT has three machine tools, it can operate in three different configurations that are described as A, B, C,

whereby the active configuration depends on the rotational position of the machine head. In each configuration, the same operation is performed in order to run the respective machine tool. This operation is modeled by the automation $G_{process}$ in Figure 2.6. When start of operation is requested with the event `ma_start_SW`, the operation of the machine tool is initiated (`ma_on`). If the completion of processing is acknowledged (`ma_ack`), the machine tool stops its operation (`ma_off`). The events of this operational sequence are summarized in Table 2.3.

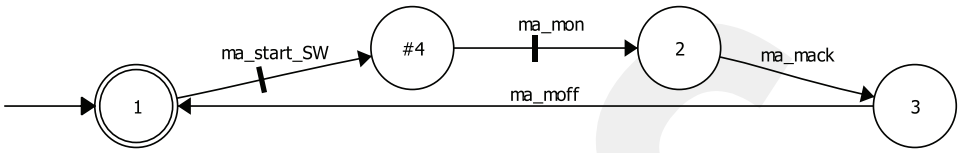

Figure 2.6: $G_{process}$

Table 2.3: Machine tools events

EVENTS NAMES	DESCRIPTION	STATUS
`ma_start_SW`	start the operation	C
`ma_mon`	switch motor on	C
`ma_mack`	finish the operation	unC
`ma_moff`	switch motor off	unC

2.1.4 Uncontrolled RMT Model

The overall uncontrolled model of the RMT is computed as the synchronous composition of all components that introduced before: G_{Belt}, $G_{vertical}$, $G_{rotation}$, and $G_{process}$. We write G_{RMT} and compute

$$G_{RMT} = G_{Belt}||G_{vertical}||G_{rotation}||G_{process} \tag{2.1}$$

Since this automaton is too big for representation in this thesis, we only provide the basic statistics. G_{RMT} has 4680 states, 27 events, and 20376 transitions. Furthermore, to achieve the desired behavior of the RMT, a supervisor should be computed and this supervisor will depend on the desired specification as introduced next.

2.1.5 RMT Supervisor

The basic desired operation of the RMT can be verbally described as follows. If a product enters the RMT, it can be processed by one of the machine tools and then leave the RMT after processing is finished. At the same time, certain safety specifications should be obeyed such as not moving the conveyor belt during processing. We next formulate the RMT behavior specification as the composition of multiple small component specifications, each of which will realize one part of the overall task. These specifications are modeled by seven finite state automata denoted as C_i with $i = 1, ..., 7$. Then, the overall specification C_{RMT} is the synchronous composition of all the small specification automata:

$$C_{RMT} = \|_{i=1}^{7} C_i \tag{2.2}$$

The specifications are listed as follows:

- *When the conveyor belt is moving, then the machine head should be standing in the upper position and all machine tools should be non-operational.* The corresponding specification automaton C_1 is shown in Figure 2.7 (a).
- *Only if there is a product on the conveyor belt, then the machine head is allowed to move to the lower position.* The corresponding specification automaton C_2 is shown in Figure 2.7 (c).
- *Whenever the machine head moves from the upper position to the lower position a tool must operate. The conveyor belt must not move while the machine head is not in the upper position.* The corresponding specification automation C_3 is shown in Figure 2.7 (b).
- *The machine head should not move while a machine tool is operating.* The corresponding specification automaton C_4 is shown in Figure 2.7 (d).
- *When the machine head is turning, the conveyor belt should not move.* The corresponding specification automaton C_5 is shown in Figure 2.7 (e).
- *The machine head is allowed to turn if the conveyor belt is empty and does not move.* The corresponding specification automaton C_6 is shown in Figure 2.7 (f).

- *If any operation is requested, then the machine tool start to operate and only after the operation is finished, the machine head can be allowed to move to the upper position.* The corresponding specification automaton C_7 is shown in Figure 2.7 (g).

Finally, the computation of the RMT supervisor S_{RMT} can be done by applying the *SupCon algorithm*, for the plant G_{RMT} and the uncontrollable events Σ_u such that

$$L_m(S_{RMT}) = SupC(C_{RMT}, L(G_{RMT}), \Sigma_u) \tag{2.3}$$

The computed supervisor has 78 states and is nonblocking according to Section 1.4. In the next step, it is possible to determine an abstraction of S_{RMT} as described in Section 1.6. This also can be done by applying the natural projection to the correct shared events $\hat{\Sigma} = \{\texttt{R-ma\_SW}, \texttt{ma-R\_SW}, \texttt{L-ma\_SW}, \texttt{ma-L\_SW}, \texttt{ma\_start\_SW}, \texttt{ma-A\_SW}, \texttt{ma-B\_SW}, \texttt{ma-C\_SW}\}$ with the neighboring components. The resulting abstracted automaton is shown in Figure 2.8.

It can be verified that the natural projection is a natural observer and locally control consistent for S_{RMT} and Σ_u. That is, the abstracted automaton $\hat{S}_{RMT}$ in Figure 2.8 can be used for a further supervisor design for the RMT and will lead to a nonblocking and maximally permissive result according to Section 1.6.

2.2 RECONFIGURATION CONTROL FOR THE RMT

As we presented in the previous section, the RMT contains three different machine tools, that can perform three different operations. In a reconfigurable manufacturing system, usually one of the tools will be used in a single configuration and changes between tools are required whenever a different configuration is requested. Hence, we now determine a reconfiguration supervisor R_{RMT} for the RMT with the following objectives:

- R_{RMT} handles changes between configurations internally.
- To the outside, R_{RMT} always performs the same operation. The first two objectives will help reducing the state size of R_{RMT}.
- R_{RMT} is composed of one modular component for each configuration. This objective will help adding new configurations if required.

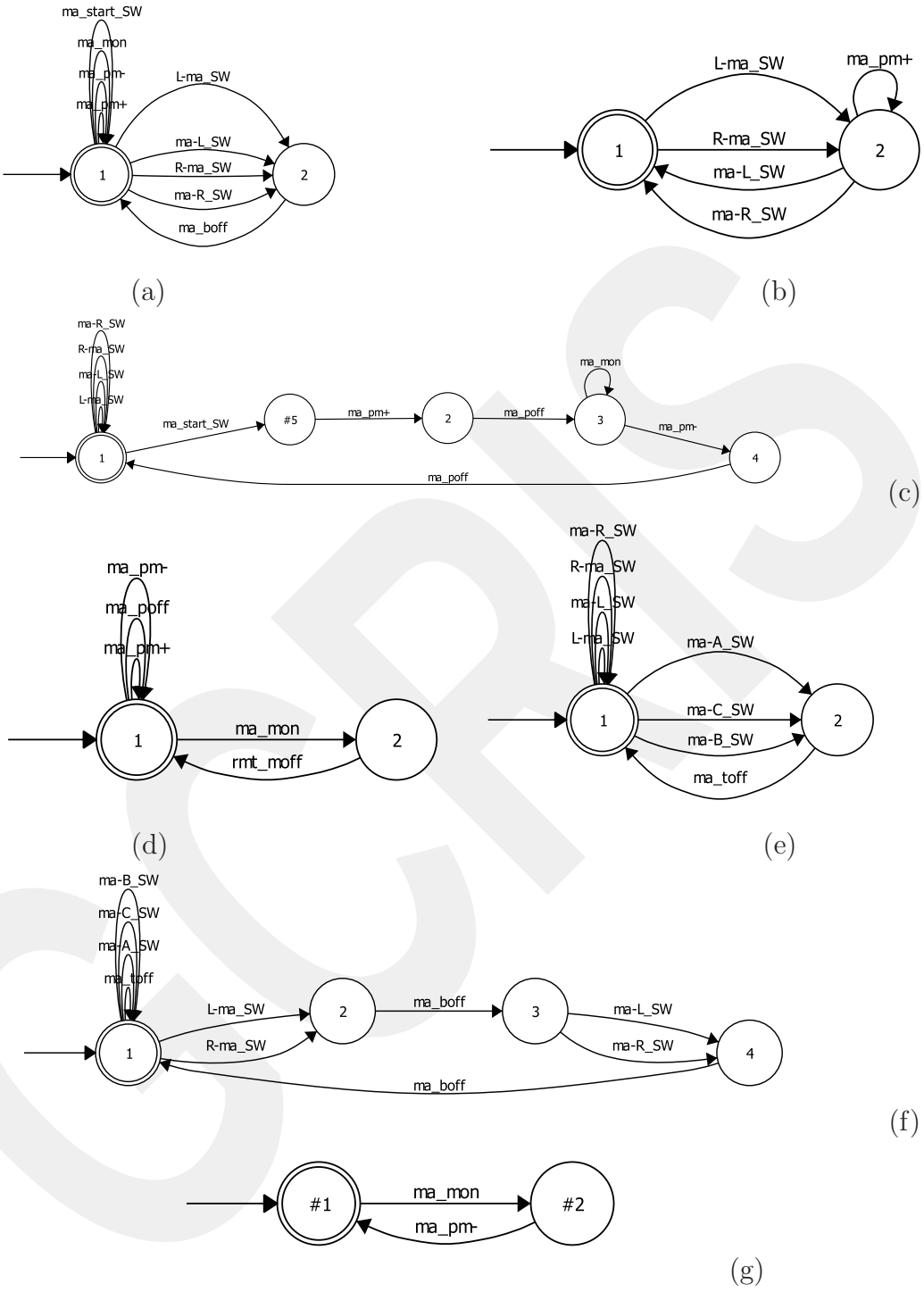

Figure 2.7: Specification automata: (a) C_1, (b) C_3, (c) C_2, (d) C_4, (e) C_5, (f) C_6, (g) C_7

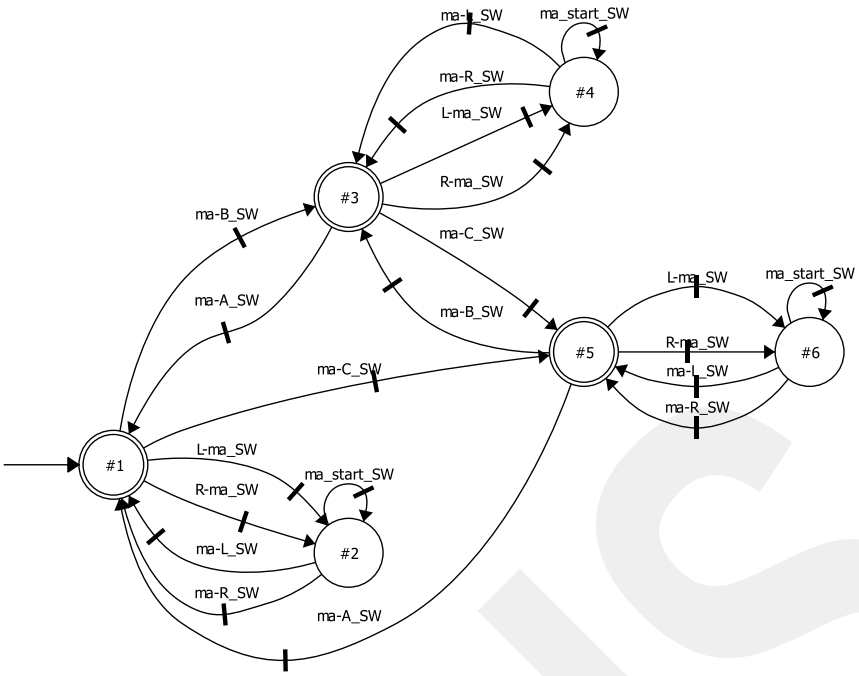

Figure 2.8: Abstraction $\hat{S}_{RMT}$ of the RMT.

Consider the high level supervisor for the RMT $\hat{S}_{RMT}$ as computed in the previous section. We use this supervisor as the plant model for the further RMT supervisor design.

In general, we use the notation $G = (X, \Sigma, \delta, x_0, X_\mathrm{m})$ for the plant automaton and identify $G = \hat{S}_{RMT}$ for the further discussion. In addition, we use the set of uncontrollable events Σ_u and denote different configurations of the RMT by a *set of configurations* $C = \{1, ..., m\}$. That is, each individual configuration can be denoted by j s.t. $j \in C$. We introduce the start state for each configuration by x^j_st, whereby configuration j should start its operation from x^j_st. Next, we divide each reconfiguration process into three parts as shown in Figure 2.9.

Our reconfiguration process for each configuration $j \in C$ consists of the active operation of j, the completion of j and the inactive state of j whenever another configuration is active. In order to change between the three parts of each configuration, we introduce reconfiguration events for each configuration j as follows:

- Reconfiguration *request* event j_req: this event can happen if another configuration is active and the activation of configuration j is requested. We write $\Sigma_\mathrm{req} = \bigcup_{j \in C} j_\mathrm{req}$.
- Reconfiguration *finish* event j_fin: this event happens if the operation of

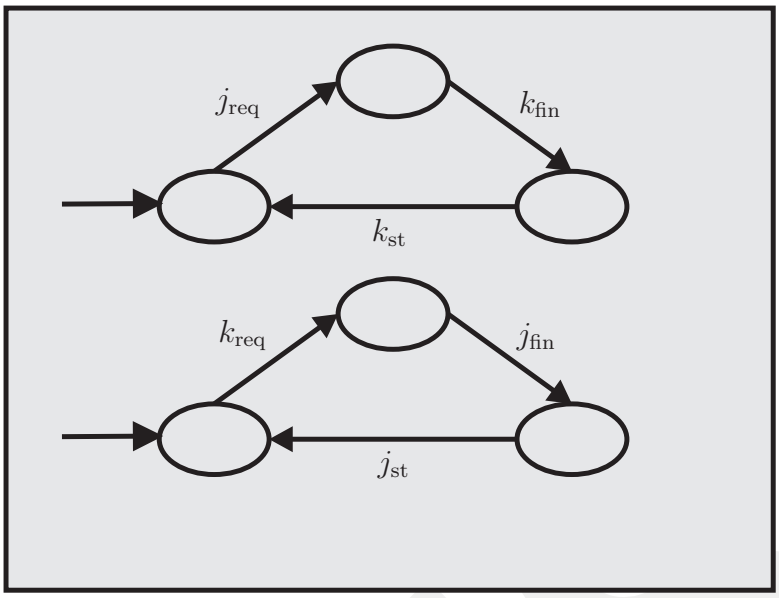

Figure 2.9: Reconfiguration process.

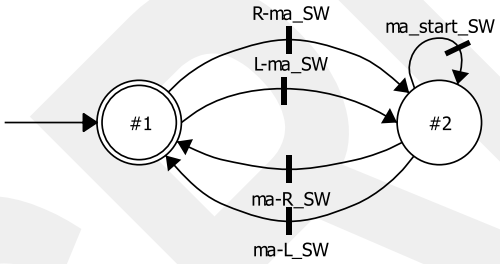

Figure 2.10: Configuration supervisor S^{act}.

configuration j is completed after another configuration was requested. We write $\Sigma_{\text{fin}} = \bigcup_{j \in C} j_{\text{fin}}$.

- Reconfiguration *start* event j_{st}: this event happens if the operation of configuration j can start. We write $\Sigma_{\text{st}} = \bigcup_{j \in C} j_{\text{st}}$.

The set of all events is then $\Sigma^{\text{rec}} = \Sigma \bigcup \Sigma_{\text{req}} \bigcup \Sigma_{\text{st}} \bigcup \Sigma_{\text{fin}}$. Using the notation introduced above, we compute the modular reconfiguration supervisor $R^{\text{j}}_{\text{RMT}} = (Z^{\text{j}}, \Sigma^{\text{rec}}, \nu^{\text{j}}, z^{\text{j}}_0, Z^{j}_{\text{m}})$, for each $j \in C$. First, we determine a supervisor for the active operation of each configuration. We denote this supervisor as $S^{\text{act}} = (Q^{\text{act}}, \Sigma, \alpha^{\text{act}}, q^{\text{act}}_0, Q^{\text{act}}_{\text{m}})$. It is important to note that we determine S^{act} such that it is identical for each configuration as discussed in Section 2.1. S^{act} for our example RMT is shown in Figure 2.10.

The modular reconfiguration supervisor R^j_{RMT} will follow S^{act} if configuration j is active. In addition to S^{act}, each modular reconfiguration supervisor R^j_{RMT} will contain the plant automaton G and one copy of the plant automaton $G' = (X', \Sigma, \delta', -, -)$. R^j_{RMT} will follow these copies if the configuration becomes inactive. Finally, we introduce *start automata* for the start of each configuration $j \in C$. The automaton $F^{\text{j,k}} = (V^{\text{j,k}}, \Sigma, \xi^{\text{j,k}}, v_0^{\text{j,k}}, V_{\text{m}}^{\text{j,k}})$. s.t. $j \in C$ and $k \in C \setminus \{j\}$ moves the plant to the start state x^j_{st} of configuration j whenever configuration k is completed.

Based on the notation introduced above, it is now possible to formulate an algorithm that computes the modular reconfiguration supervisor R^j_{RMT}.

NEW ALGORITHM
We compute the modular reconfiguration supervisor $R^{\text{j}}_{\text{RMT}} = (Z^{\text{j}}, \Sigma^{\text{rec}}, \nu^{\text{j}}, z_0^{\text{j}}, Z^j_{\text{m}})$, for each configuration $j \in C$ and $j = 1, ..., m$.

- **Input:** plant automaton $G = (X, \Sigma, \delta, x_0, X_{\text{m}})$, copy of plant automaton $G' = (X', \Sigma, \delta', -, -)$, supervisor automaton $S^{\text{j}} = (Q^{\text{j}}, \Sigma, \alpha^{\text{j}}, q_0^{\text{j}}, Q_{\text{m}}^{\text{j}})$, and start configuration automaton $F^{\text{j,k}} = (V^{\text{j,k}}, \Sigma, \xi^{\text{j,k}}, v_0^{\text{j,k}}, V_{\text{m}}^{\text{j,k}})$.

$$Z^{\text{j}} = Q^{\text{j}} f \bigcup X \bigcup X' \bigcup V^{\text{j,k}} \tag{2.4}$$

$$Z^{\text{j}}_{\text{m}} = Q^{\text{j}}_{\text{m}} \bigcup X_{\text{m}} \bigcup V^{\text{j,k}}_{\text{m}} \tag{2.5}$$

$$z_0^{\text{j}} = q_0^{\text{j}} \text{ for } j = 1 \text{ otherwise } z_0^{\text{j}} = x^1_{\text{st}} \tag{2.6}$$

For each $q \in Q^{\text{j}}$ and $\sigma \in \Sigma$

$$\alpha^{\text{j}}(q, \sigma)! \rightarrow \nu^{\text{j}}(q, \sigma) = \alpha^{\text{j}}(q, \sigma) \tag{2.7}$$

For each $q \in Q^{\text{j}}$ and $\sigma \in \Sigma_{\text{req}} \setminus \{j_{\text{req}}\}$

$$\nu^{\text{j}}(q, \sigma) = q \tag{2.8}$$

For each $x \in X$ and $\sigma \in \Sigma$

$$\delta(x, \sigma)! \rightarrow \nu^{\text{j}}(x, \sigma) = \delta(x, \sigma) \tag{2.9}$$

For each $x' \in X'$ and $\sigma \in \Sigma$

$$\delta'(x', \sigma)! \rightarrow \nu^{\text{j}}(x', \sigma) = \delta'(x', \sigma) \tag{2.10}$$

For each $q \in Q^{\mathrm{j}}$ s.t. $q = q_0^{\mathrm{j}}$,$\sigma = j_{\mathrm{fin}}$ and $x = x_{\mathrm{st}}^1 \in X$

$$\nu^{\mathrm{j}}(q, \sigma) = x \tag{2.11}$$

For each $x \in X$, $x' \in X'$ and $\sigma \in \Sigma_{\mathrm{req}}$ when $j = 1$

$$\nu^{\mathrm{j}}(x, \sigma) = x' \tag{2.12}$$

For each $x' \in X'$ with $j \neq 1$ and $\sigma \in \Sigma_{\mathrm{fin}} \setminus \{j_{\mathrm{fin}}\}$

$$\nu^{\mathrm{j}}(x', \sigma) = v_0^{\mathrm{j,k}} \tag{2.13}$$

For each $v \in V^{\mathrm{j,k}}$ and $\sigma \in \Sigma$

$$\xi^{\mathrm{j,k}}(v, \sigma)! \rightarrow \nu^{\mathrm{j}}(v, \sigma) = \xi^{\mathrm{j,k}}(v, \sigma) \tag{2.14}$$

For $v \in V_m^{\mathrm{j,k}}$ and $\sigma = j_{\mathrm{st}}$

$$\nu^{\mathrm{j}}(v, \sigma) = z_0^{\mathrm{j}} \tag{2.15}$$

In words, the function of the reconfiguration supervisor $R_{\mathrm{RMT}}^{\mathrm{j}}$ can be summarized by *"Perform the task of configuration j if it is active and switch to another configuration if it requested after completing the active configuration j"*. Each part of the reconfiguration supervisor will take care of one of the described tasks. The supervisor S^{act} is used when the configuration $j \in C$ is active. If another configurations is requested, R_{RMT}^j keeps following S^{act} until the configuration finish event j_{fin} occurs. Then, it moves to the plant automaton G and follows it until configuration j is requested again with j_{req}. In that case, R_{RMT}^j switches to the copy of the plant G' and follows this copy until a state is reached where the currently active configuration k is completed. Then, it is only required to move to the configuration start state x_{st}^j in order to start configuration j, which is performed by the automaton $F^{\mathrm{j,k}}$. After $F^{j,k}$ completes its operation, configuration j is started with j_{st} and R_{RMT}^j moves to S^{act}. The basic procedure is shown in Figure 2.11.

The result of the above computation is one modular reconfiguration supervisor R_{RMT}^j for each configuration $j \in C$. The overall reconfiguration supervisor of the RMT is hence given by the synchronous composition of these modular reconfiguration supervisors as

$$R_{RMT} = ||_{j \in C} R_{RMT}^j. \tag{2.16}$$

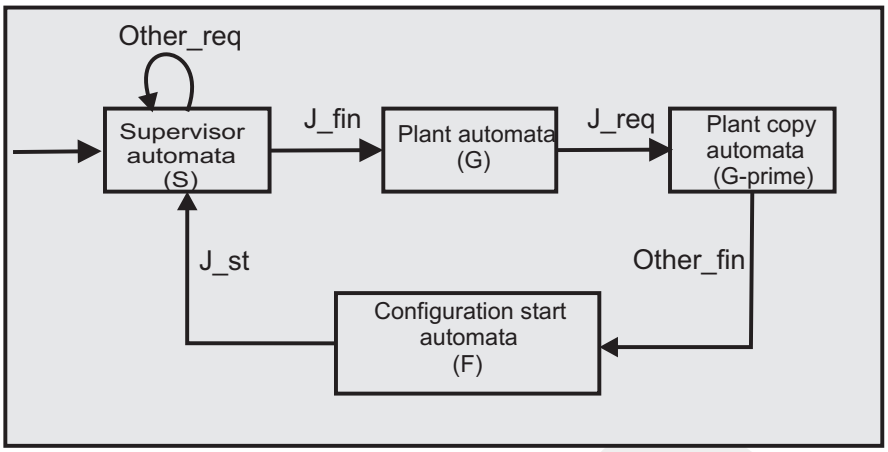

Figure 2.11: Reconfiguration overview

The resulting reconfiguration supervisors R^{A}_{RmT} and R^{B}_{RmT} are shown in Figure 2.12 and 2.13.

Furthermore, we recall the notation of the abstraction-based control and abstract the result of our composition. The high level supervisor is found identical to the RMT operation in each configuration in Figure 2.10 with only two states. That is, our RMT supervisor can be conveniently used for a further design.

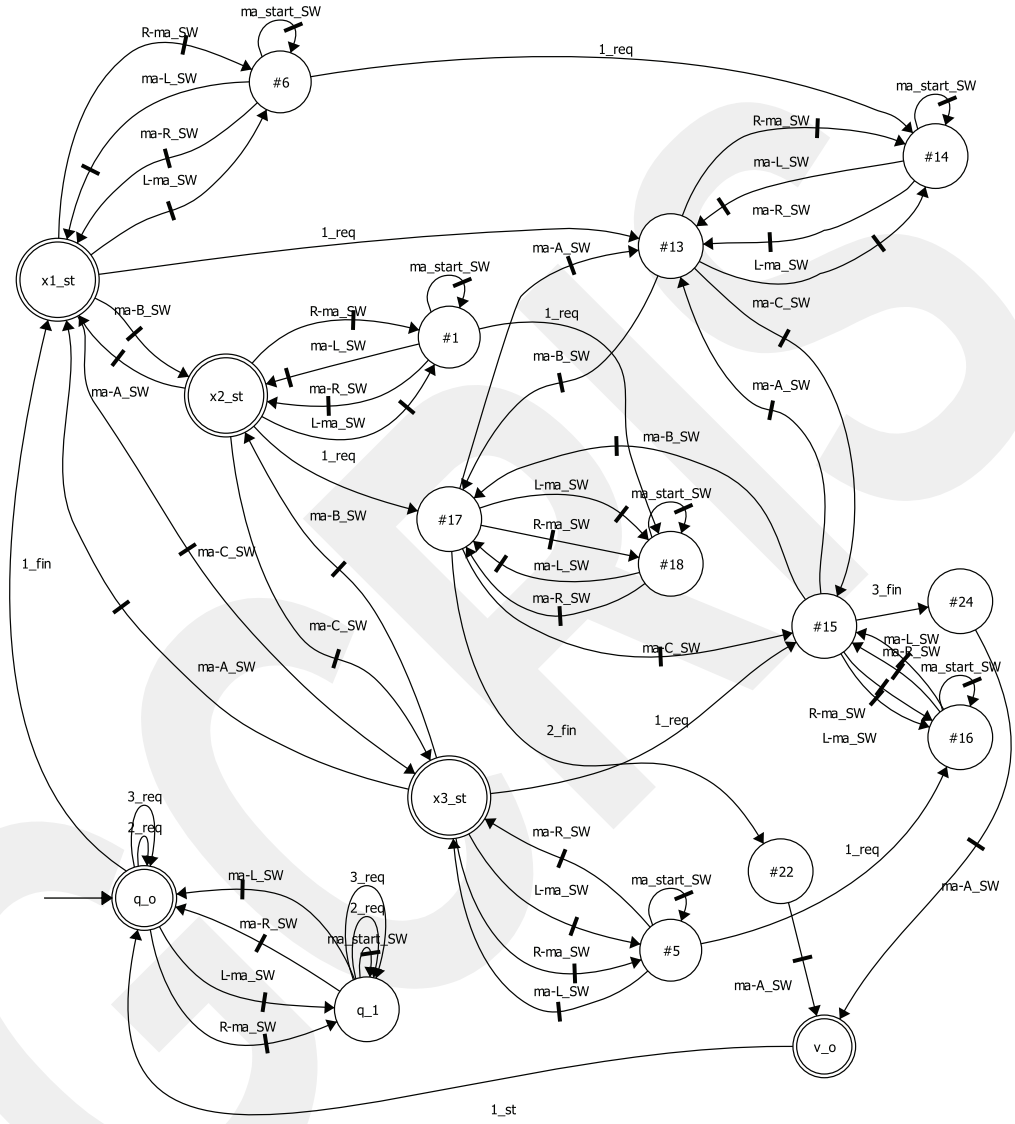

Figure 2.12: R^A_{RMT}

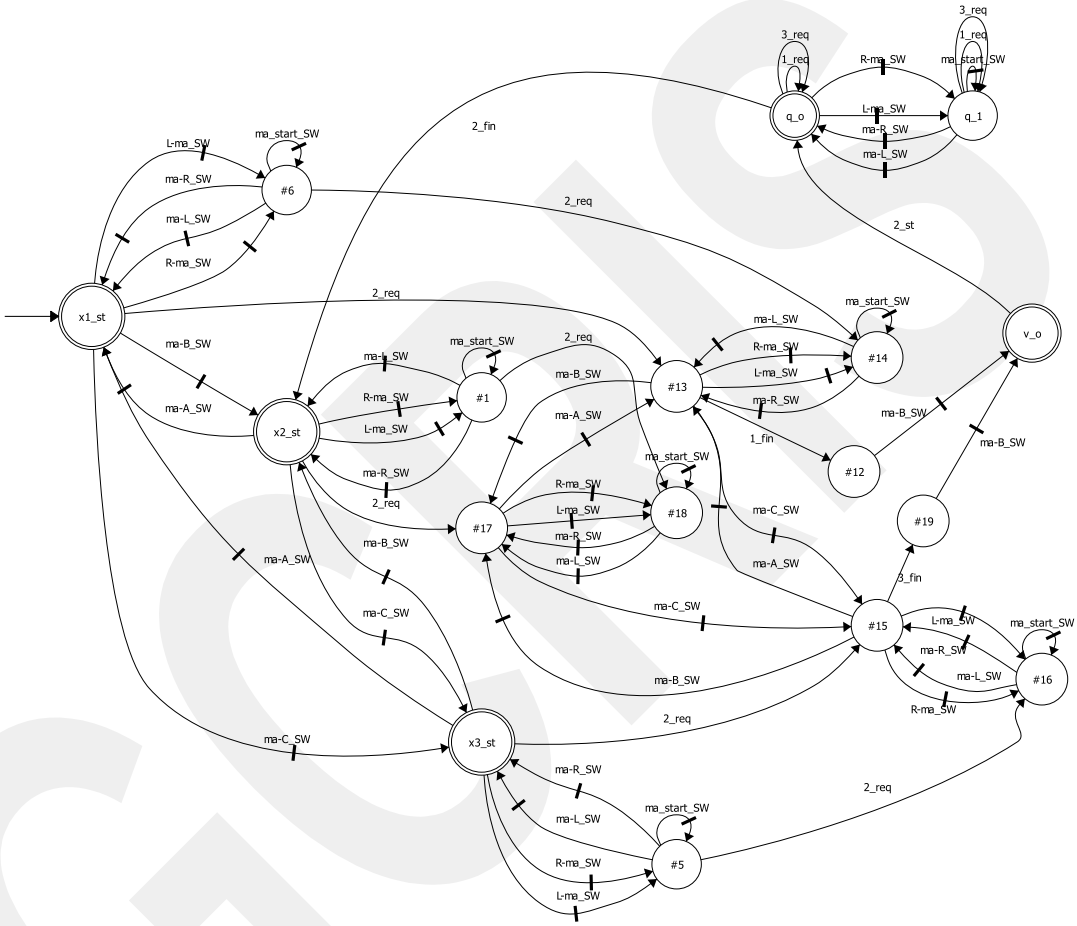

Figure 2.13: R^B_{RMT}

CHAPTER III

ABSTRACTION-BASED RECONFIGURATION CONTROL FOR RECONFIGURABLE MANUFACTURING SYSTEMS (RMS)

In these days, we know that the manufacturing industry can be impacted by the customers and market demands. The requirement to produce a large variety products in a short time at a reasonable cost, drive these industries to invent and develop new technologies. Reconfigurable manufacturing systems (RMS) are targeted to enable these manufacturers to improve the capacity and the quality of production, and produce a large variety products with less cost and time.

The concept of *reconfiguration* can be defined as how to produce different products and product types with the same machine components of a manufacturing systems. In this context, reconfiguration control is needed to change the active operation of the system.

In this thesis, we develop the idea of the reconfiguration control for large-scale RMS. This idea is based on the modular controller design which is possible since RMS are usually composed of multiple components including RMTs.

The design of reconfiguration control is subject to the following requirements:

- In each configuration, the desired behavior for the mode of operation should be achieved.
- If a new configuration is requested, then the switching to the requested configuration should be performed as fast as possible.
- The design should be capable to handle large scale systems.

In this chapter, we introduce a new approach for the design of reconfiguration supervisors that is based on the computation of modular configuration supervisors and state attraction supervisors (used to complete each active configuration). We

develop the different steps of the computation using an illustrative example that is described in Section 3.1. Then, we explain the supervisor computation first for a single RMS module in Section 3.2. Then, we show how abstraction-based supervisory control can be used in order to achieve nonblocking control for large-scale RMS in Section 3.3. The presented results are obtained based on the previous results in [9, 16, 21, 24, 23]

3.1 ILLUSTRATIVE EXAMPLE SYSTEM

3.1.1 Example Overview

Our RMS example system consists of different manufacturing components as shown in Figure 3.1. We first introduce the models of these components separately, and then we separate the RMT into two modules.

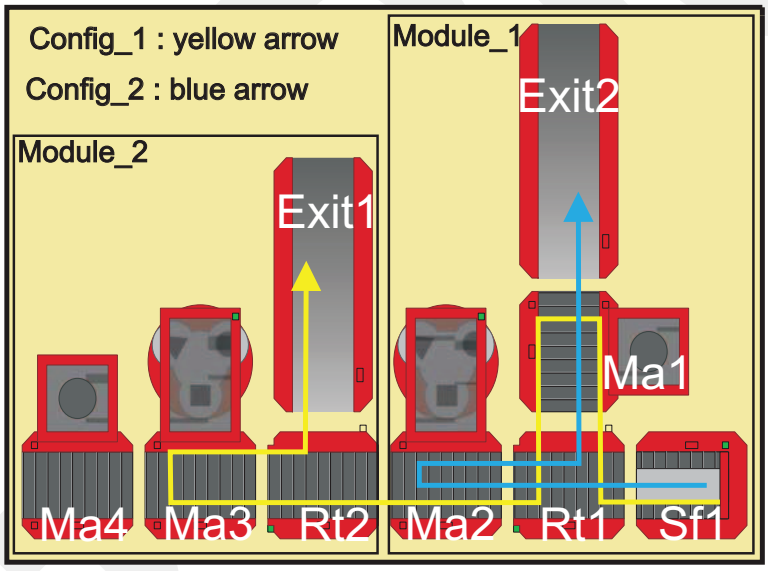

Figure 3.1: Overview RMS example

The components of the example RMS are

- One stack feeder SF1
- Two rotary tables RT1 and RT2
- Two RMTs MA2 and MA3
- Two single production machines MA1 and MA4

We consider two configurations of the RMS that are described as follows.

Configuration (1):

Products enter the system from SF1 and are transported to MA1 for processing by the rotary table RT1. Then, products are moved back to RT1 and are transported to MA2 for processing. From MA2, products are transported to RT2 and to MA3. Processing is performed by MA3 and then products are moved out by RT2.

Configuration (2):

Products enter from SF1 and are moved to RT1 and then MA2 for processing. Then, products are moved back by RT1 and MA1. After processing in MA1, the products are transport out of the RMS from MA1.

Since the RMS consists of multiple manufacturing components, we divide the overall system into two modules. These modules are described in the sequel.

3.1.2 Module 1

We consider the subpart of the RMS in Figure 3.2 as module 1. This module consists of one stack feeder (SF1), one rotary table (RT1), one RMT denoted by (MA2), and one single production machine (MA1). The exits slides Exit1 and Exit2 symbolize the output of products from the considered module. We next

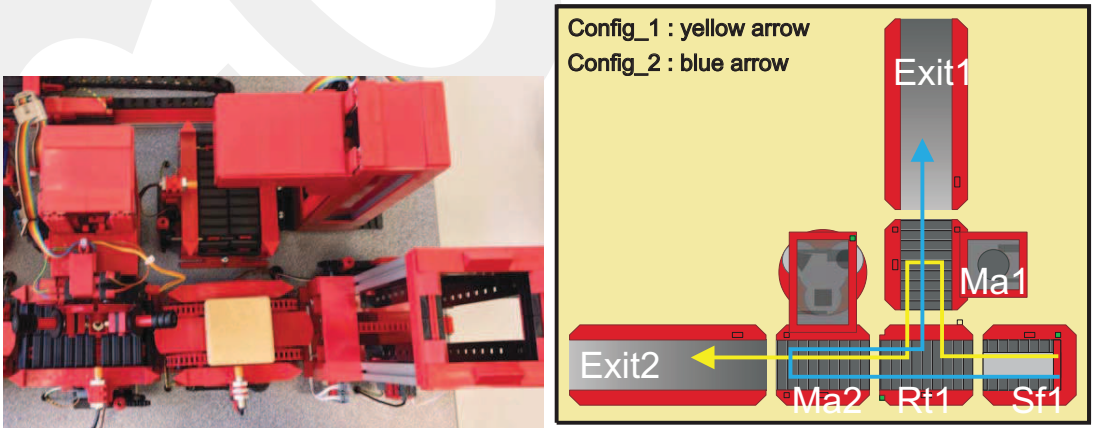

Figure 3.2: Module 1: picture of the physical system (left) and schematic (right)

model all of the system components by automata and then compute the overall model of the module by the synchronous composition of its components models.

Stack Feader (SF1)

Products can enter the system by SF1. The stack feeder moves unprocessed products to the neighbor component which is RT1 in our example. This hardware component shown in Figure 3.3 consist of many parts. First, a stack feeder tower holding products. Second, a conveyor belt with a block to push products and move them to the neighbor. Third, a magnetic sensor to detect the position of the block, and a motor that turns the belt. Table 3.1 shows the explanations for all events that are used for modeling of SF1.

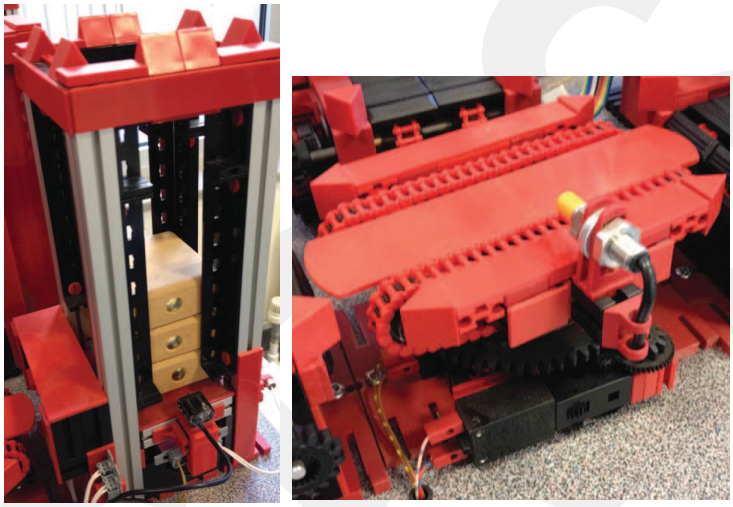

Figure 3.3: Picture of the stack feeder (left) and rotary table (right)

Table 3.1: Stack feader events

EVENTS NAMES	DESCRIPTION	STATUS
`sf1-rt1_SW`	move products (SF1 to RT1)	C
`sf1_fdon`	SF on	C
`sf1_fdoff`	SF off	C
`sf1_fdhome`	SF in home position	unC
`sf1_wpar`	sensor detects products arrives	unC
`sf1_wplv`	sensor detects products leaves	unC
`sf1_lvtort1`	products leaves to (RT1)	unC
`rt1_lvtosf1`	products leaves to (SF1)	unC

The plant behavior of SF1 is modeled by the automaton G_{SF1} in Figure 3.4 and an abstraction of this DES plant is computed according to Section 1.6. Hereby, set of shared events include the event (`sf1-rt1_SW`), because this event is the only

shared event between SF1 and its neighbor RT1. The resulting abstraction $G_{\text{SF1}}^{\text{high}}$ is shown in Figure 3.5.

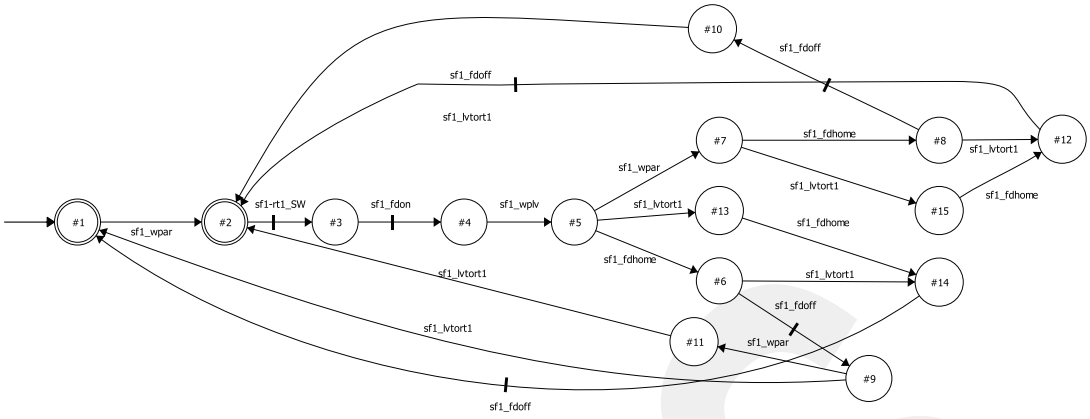

Figure 3.4: G_{SF1}

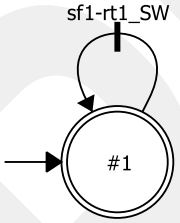

Figure 3.5: $G_{\text{SF1}}^{\text{high}}$

Rotary Table (RT1)

Figure 3.3 shows the picture of the actual rotary table. It can rotate in two directions, either clockwise or counter clockwise, and stop at a horizontal or vertical position. Accordingly, RT1 can receive and deliver products in four directions that are UP, DOWN, LEFT, RIGHT. From the Figure of the RMS example we can notice that RT1 is located between three components, SF1 from the right, MA2 from the left, and MA1 in the upper position. We denote the plant automaton for RT1 as G_{RT1}. Since this automaton contains a large number of states, (38 states, 22 events, and 50 transitions), we only show the high level automaton that will be used in the overall plant of the RMS. In Table 3.2, all events that describe the behavior of RT1 are listed, and the abstracted automaton $G_{\text{RT1}}^{\text{high}}$ is depicted in Figure 3.3.

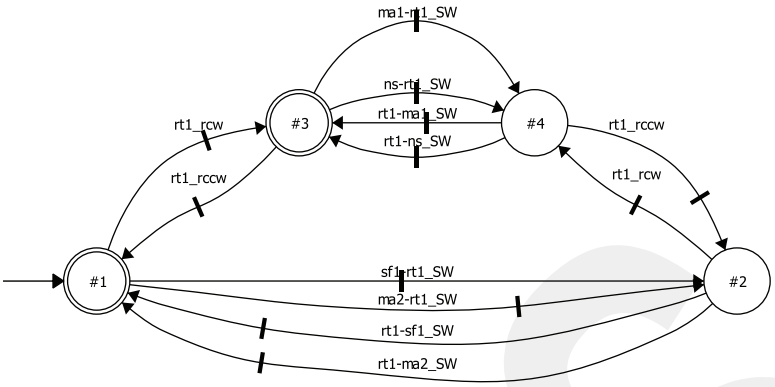

Figure 3.6: $G^{\text{high}}_{\text{RT1}}$

Table 3.2: Rotary table events

EVENTS NAMES	DESCRIPTION	STATUS
sf1-rt1_SW	move products (SF1 to RT1)	C
rt1-sf1_SW	move products (RT1 to SF1)	C
ma2-rt1_SW	move products (MA2 to RT1)	C
rt1-ma2_SW	move products (RT1 to MA2)	C
ma1-rt1_SW	move products (MA1 to RT1)	C
rt1-ma1_SW	move products (RT1 to MA1)	C
ns-rt1_SW	move products (NS to RT1)	C
rt1-ns_SW	move products (RT1 to NS)	C
rt1_rcw	turns RT1 (clockwise)	C
rt1-_rccw	turns RT1 (counter clockwise)	C

Manufacturing Production Machine (MA1)

The production machine contains *conveyor belt*, *machine head*, and one single *machine tool*. For this reason, the components models and the overall model of MA1 is almost the same as the overall model for the RMT as discussed in section 2.1. The only difference is that there is no rotation movement, because there is just one machine tool. The overall model for the MA1is equal to the parallel composition of these components models, denoted by $G_{\text{MA1}} = G_{Belt}||G_{vertical}||G_{process}$, as introduced before. G_{MA1} is shown in Figure 3.7, and its abstraction $G_{\text{MA1}}^{\text{high}}$ is depicted in Figure 3.8. Hereby, all related events are summarized in Table 3.3. MA1 is located between RT1 and CO15.

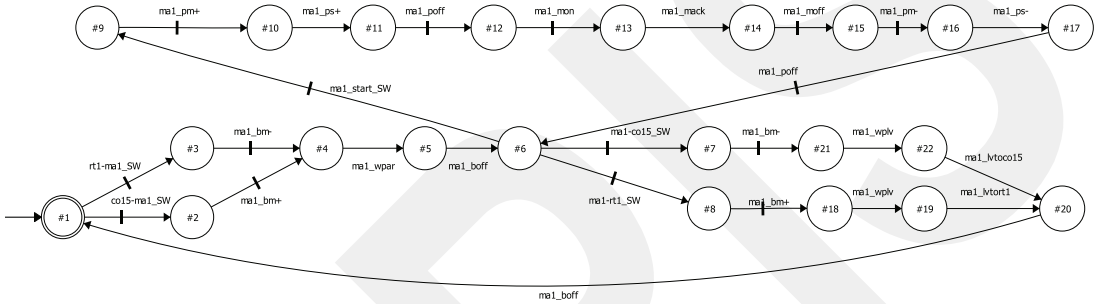

Figure 3.7: G_{MA1}

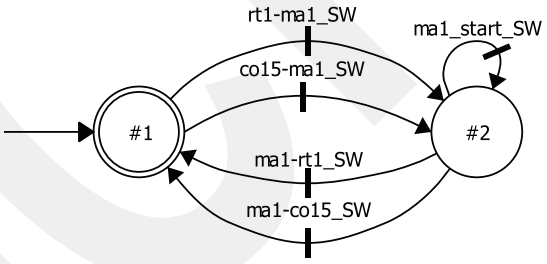

Figure 3.8: $G_{\text{MA1}}^{\text{high}}$

Reconfigurable Machine Tool RMT (MA2)

In section 2.1, we introduced the general model of the RMT. The same model is used here with the neighbor components RT1 at the right position and RT2 at the left. The plant automaton $G_{\text{MA2}}^{\text{high}}$ is shown in in Figure 3.9.

The overall plant model for the RMS is denoted as G_{RMS1} and is computed as the synchronous composition of SF1, RT1, MA1, and MA2. The result is too big to

Table 3.3: Production machine events

EVENTS NAMES	DESCRIPTION	STATUS
rt1-ma1_SW	move products (RT1 to MA1)	C
ma1-rt1_SW	move products (MA1 to RT1)	C
co15-ma1_SW	move products (CO15 to MA1)	C
ma1-co15_SW	move products (MA1 to CO15)	C
ma1_start_SW	start the operation of (MA1)	C

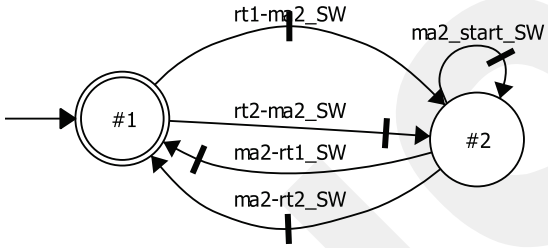

Figure 3.9: $G^{\text{high}}_{\text{MA2}}$

be shown, (48 states, 19 events, and 312 transition).

$$G_{\text{RMS1}} = G^{\text{high}}_{\text{SF1}} || G^{\text{high}}_{\text{RT1}} || G^{\text{high}}_{\text{MA1}} || G^{\text{high}}_{\text{MA2}} \tag{3.1}$$

3.1.3 Module 2

Module 2 consists of the remaining components of our example RMS as can be seen in Figure 3.10. It comprises one rotary table (RT2), one RMT (MA3), and one single production machine (MA4).

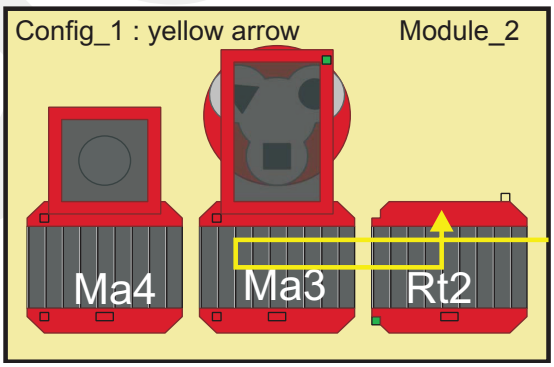

Figure 3.10: Schematic of module 2

Rotary Table (RT2)

The automaton for modeling RT2 is the same as we discussed before for RT1, with different neighbor components. Figure 3.11 shows this automaton.

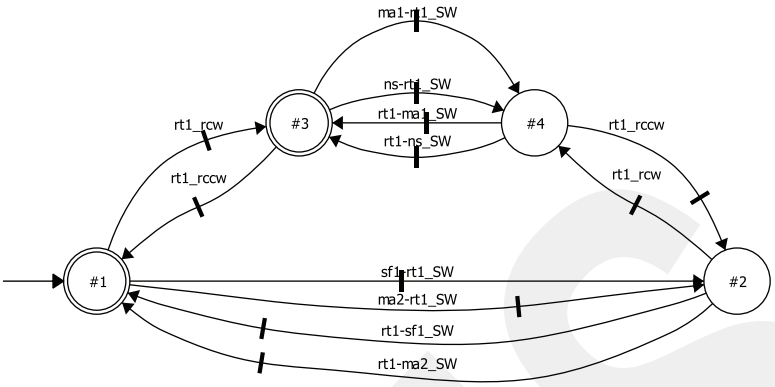

Figure 3.11: $G^{\text{high}}_{\text{RT2}}$

Reconfigurable Machine Tool RMT (MA3)

As in section 2.1, the same model is used for the RMT only with different neighbor components. The plant automaton $G^{\text{high}}_{\text{MA3}}$ is shown in Figure 3.12.

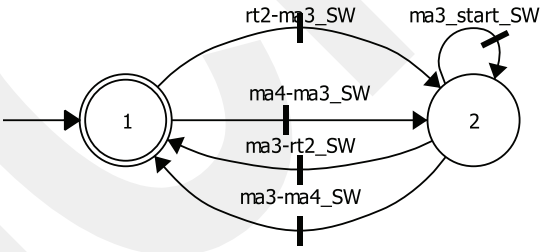

Figure 3.12: $G^{\text{high}}_{\text{MA3}}$

Single Production Machine (MA4)

The automaton $G^{\text{high}}_{\text{MA4}}$ is shown in Figure 3.13. The neighbors are MA3 from the right position and L in the left position

The overall plant model for module 2 is denoted as G_{RMS2} and is computed as the synchronous composition of RT2, MA3, and MA4. The result is too big to be

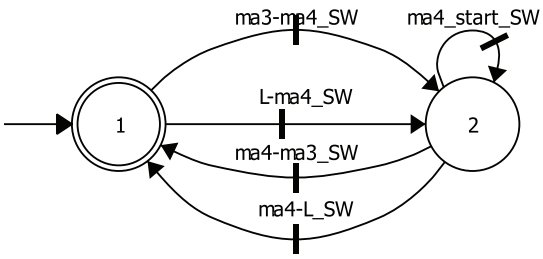

Figure 3.13: $G^{\text{high}}_{\text{MA4}}$

shown, (48 states, 19 events, and 300 transition).

$$G_{\text{RMS2}} = G^{\text{high}}_{\text{RT2}} || G^{\text{high}}_{\text{MA3}} || G^{\text{high}}_{\text{MA4}} \tag{3.2}$$

3.2 RECONFIGURATION CONTROL FOR RMS WITH ONE MODULE

3.2.1 Reconfiguration control For RMS module 1

As described before, we consider two configurations of the RMS. In the first configuration, products can enter the system from SF1, move to RT1 then to MA1. After processing in MA1, products move through RT1 to MA2, are processed in MA2 and then transported out of the system. In the second configuration, products can enter the system from SF1, move to processing in MA2 via RT1. Products then move back through RT1 to MA1 and leave the system after processing at MA1. The specification for each individual configuration can given as: $specification_1$ for $configuration_1$ will be equal to the synchronous composition of three small specifications shown in Figure 3.14, s.t. $specification_1 = C_{1,\text{con1}} || C_{2,\text{con1}} || C_{3,\text{con1}}$.

The component automata for $specification_2$ for $configuration_2$ are shown in Figure 3.15 such that

$$specification_2 = C_{1,\text{con2}} || C_{2,\text{con2}} || C_{3,\text{con2}} \tag{3.3}$$

Now, according to the classical supervisory control theory, the supervisor for each configuration with each related specification can be computed to achieve the desired operation. We obtain $supervisor_{\text{con1}}$ and $supervisor_{\text{con2}}$ such that

$$L_{\text{m}}(supervisor_{\text{con1}}) = SupC(L_{\text{m}}(specification_1), G_{\text{RMS1}}, \Sigma_{\text{u}}) \tag{3.4}$$

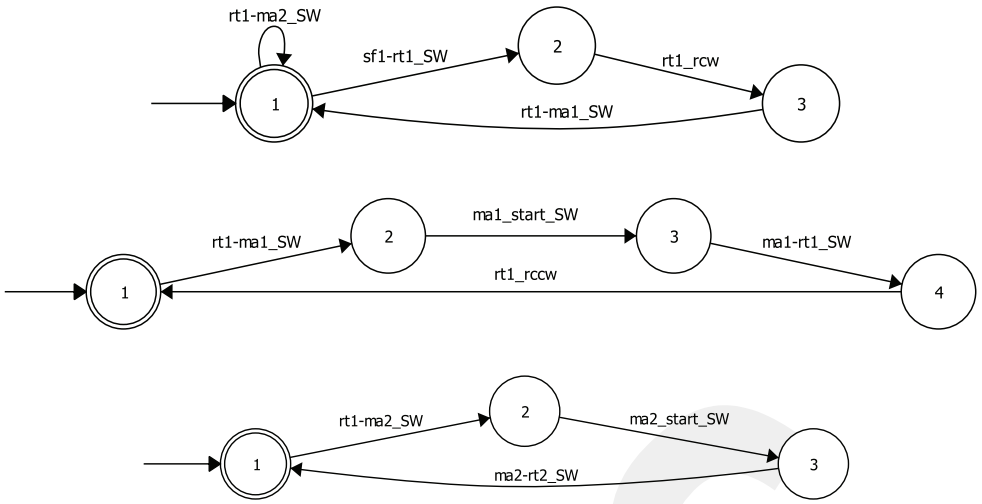

Figure 3.14: Specification 1

and

$$L_{\mathrm{m}}(supervisor_{\mathrm{con2}}) = SupC(L_{\mathrm{m}}(specification_2), G_{\mathrm{RMS1}}, \Sigma_{\mathrm{u}}) \tag{3.5}$$

The resulting automata are depicted in Figure 3.2.1. Until here, we presented the plant, the specifications, and the supervisors that represent the desired operation for each configuration. In the next step, we address the reconfiguration problem and how to design the reconfiguration supervisors for a single module of the RMS.

In order to change between configurations, we suggest to first complete the currently active configuration before starting a new configuration. This task is accomplished by computing a supervisor for state attraction (see Section 1.5) that moves the system back to its initial state. The supervisors for state attraction are too big to shown for module 1, we introduce the supervisor attractor for the second configuration as in Figure 3.17.

Next, we combine the supervisor for the desired operation of each configuration and the completion of the configuration. Hereby, we use events $conf1_{req}/conf2_{req}$ that happen if configuration 1/2 is requested, $conf1_{fin}/conf2_{fin}$ which happen if configuration 1/2 is completed and $conf1_{st}/conf2_{st}$ that happen if configuration 1/2 are started. The supervisor and state attractor are connected with these events as shown for the reconfiguration process is in Figure 3.18 and an example, the reconfiguration supervisor 1 in Figure 3.19.

That is, the reconfiguration supervisor moves from the supervisor part to the state

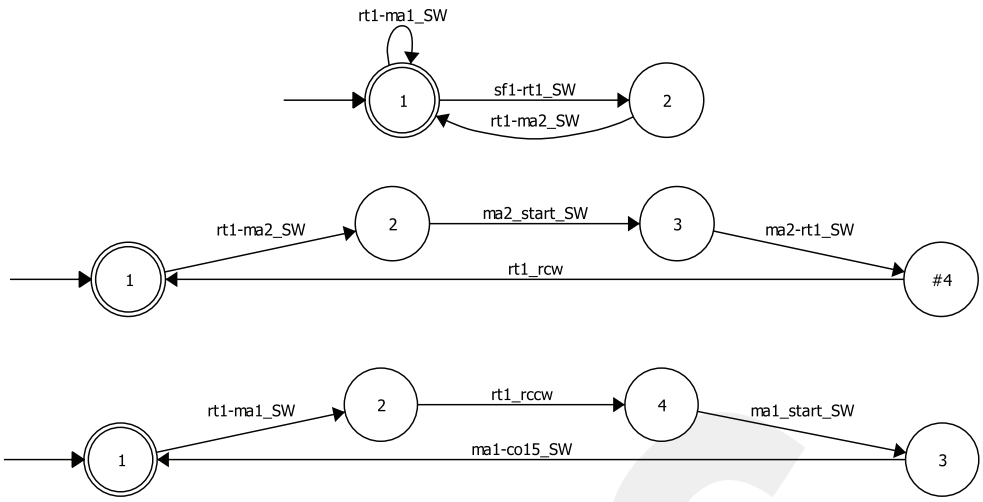

Figure 3.15: Specification 2

attractor part with the event $conf1_{req}$. This means that in the active configuration 2, configuration 1 is requested. That is configuration 2 has to be completed by moving to the initial state using the state attractor. After the initial state is reached, the event $conf2_{fin}$ happens, which indicates that the configuration 2 is completed. This event leads to a waiting state, where configuration 2 is inactive and waits to be activated. Activation is performed when the event $conf2_{st}$ happens. In this case, the operation of configuration 2 starts again. Finally, the statistics for the results are illustrated as in Table 3.4.

Table 3.4: Module 1 supervisors

SUPERVISORS	STATES	EVENTS	TRANSITIONS
$supervisor_{\text{con1}}$	21	16	33
$supervisor_{\text{con2}}$	19	16	29
R_{con1}	43	22	110
R_{con2}	39	22	100

3.2.2 General Formulation of the Reconfiguration Problem

Modular Reconfiguration Supervisors

We formalize the reconfiguration problem according to the previous discussion. We consider that the RMS is composed of n modules For each module, we intro-

Figure 3.16: Supervisor for configuration 1 (left) and configuration 2 (right)

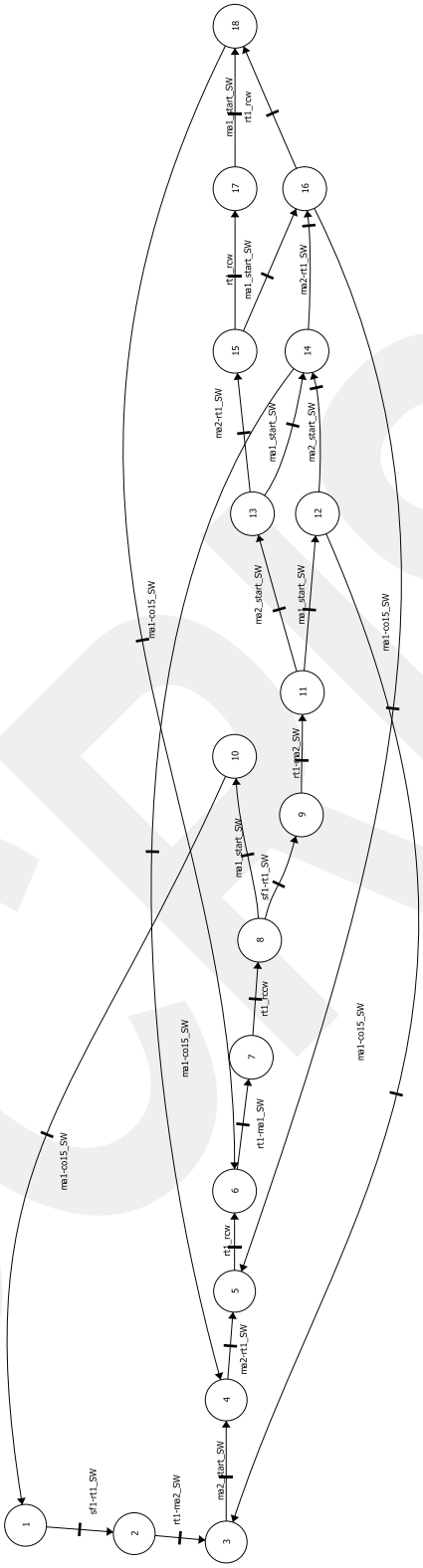

Figure 3.17: Attractor supervisor for configuration 2

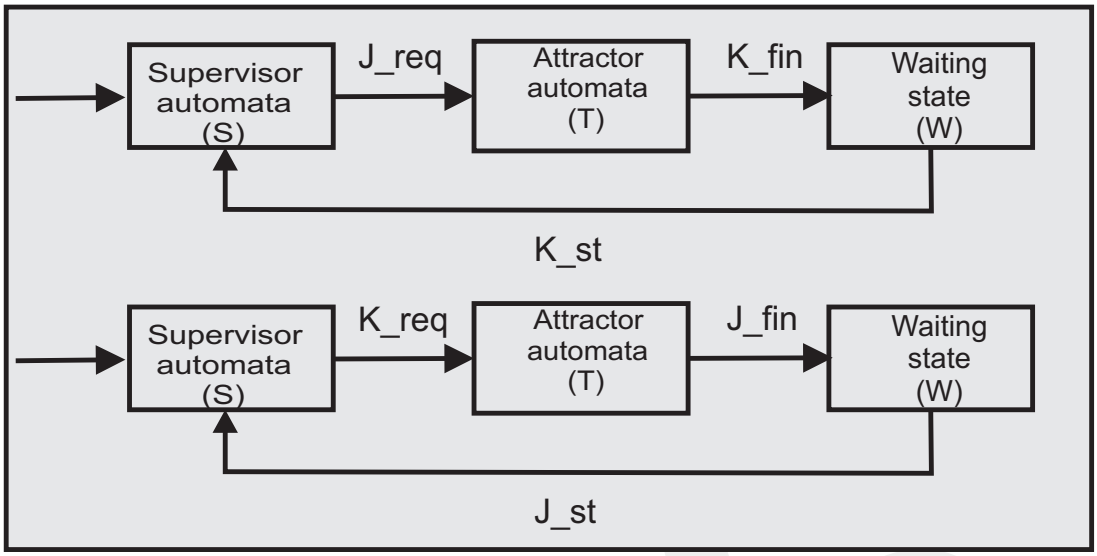

Figure 3.18: Reconfiguration supervisor process

duce a plant automaton G_i and uncontrollable events $\Sigma_{\mathrm{u},i}$ for $i = 1, \ldots, n$. We want to realize a set of m configurations $\mathcal{C} = \{1, \ldots, m\}$, whereby each configuration $j \in \mathcal{C}$ comprises component specifications $K_i^j \subseteq L_{\mathrm{m}}(G_i)$, $i = 1, \ldots, n$, and a global specification $\hat{K}^j \subseteq (\hat{\Sigma}^j)^\star$. Each configuration $j \in \mathcal{C}$ is realized by supervisor components $S_i^j = (Q_i^j, \Sigma_i, \nu_i^j, q_{0,i}^j, Q_{\mathrm{m},i}^j)$ for $i = 1, \ldots, n$. In order to model the reconfiguration, we introduce a *reconfiguration request event* j_{req}, a *reconfiguration finish event* j_{fin} and a *reconfiguration start event* j_{st} for each $j \in \mathcal{C}$. We write $\Sigma_{\mathrm{req}} = \bigcup_{j\in\mathcal{C}}\{j_{\mathrm{req}}\}$, $\Sigma_{\mathrm{fin}} = \bigcup_{j\in\mathcal{C}}\{j_{\mathrm{fin}}\}$, $\Sigma_{\mathrm{st}} = \bigcup_{j\in\mathcal{C}}\{j_{\mathrm{st}}\}$ and $\Sigma_i^{\mathrm{rec}} = \Sigma_i \cup \Sigma_{\mathrm{req}} \cup \Sigma_{\mathrm{fin}} \cup \Sigma_{\mathrm{st}}$.

Using the introduced notation, we compute the reconfiguration supervisors $R_i^j = (Z_i^j, \Sigma^{\mathrm{rec}}, \alpha_i^j, z_{0,i}^j, Z_{\mathrm{m},i}^j)$ for each module $i = 1, \ldots, n$ and each configuration $j \in \mathcal{C}$. We have Q_i^j a states set for all S_i^j, assume that automata $T_i^j = (Q_i^j, \Sigma, \omega_i^j, -, -) \sqsubseteq S_i^j$ are computed such that $\{q_{0,i}^j\}$ is a strong attractor for Q^j. This means that T_i^j is a state attractor for module i and configuration j that completes the configuration by moving to the initial state. The construction of the reconfiguration supervisor for module i and configuration j is now done as described for module 1 in the previous section. The exact computation rules are given as follows.

$$Z_i^j = Q_i^j \cup \{q' | q \in Q_i^j\} \cup \{\mathrm{W}\} \tag{3.6}$$

$$Z_{\mathrm{m}}^j = Q_{\mathrm{m}}^j \cup \{\mathrm{W}\} \tag{3.7}$$

$$z_{0,i}^j = q_{0,i}^j \text{ if } j = 1 \text{ and } z_{0,i}^j = \mathrm{W} \text{ otherwise} \tag{3.8}$$

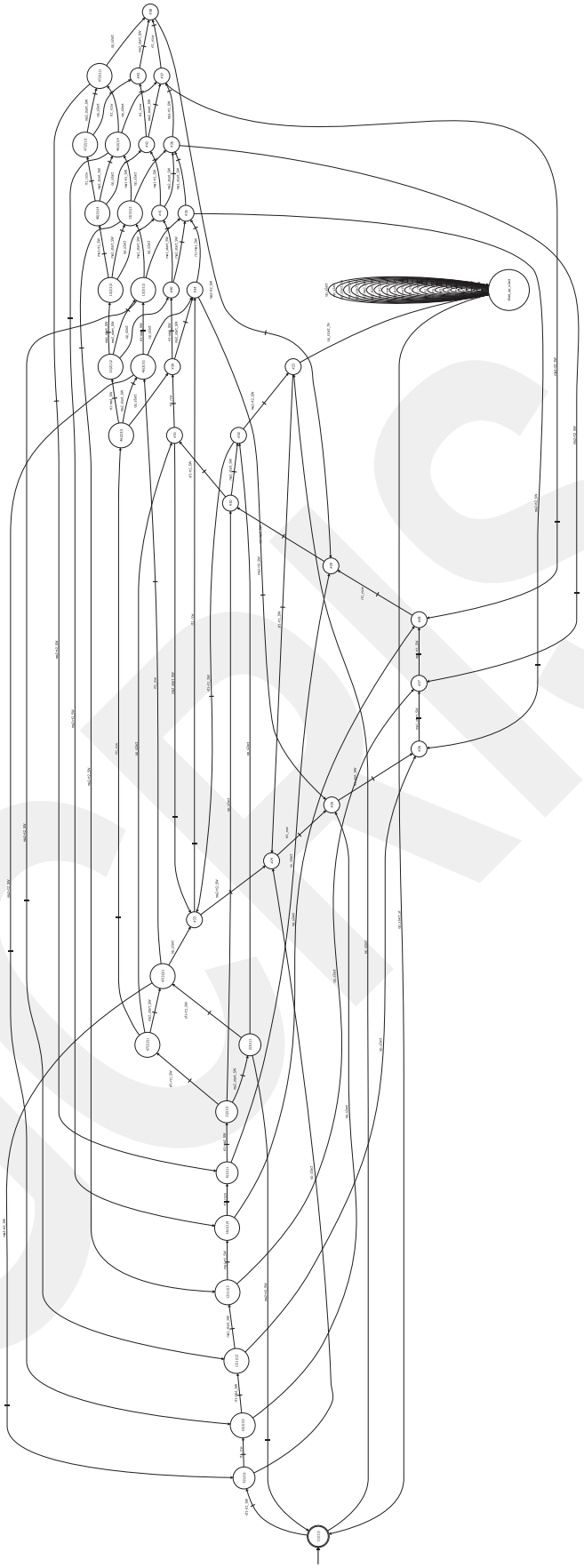

Figure 3.19: Reconfiguration supervisor for configuration 1

For each $q \in Q_i^j$ and $\sigma \in \Sigma$

$$\nu_i^j(q,\sigma)! \Rightarrow \alpha_i^j(q,\sigma) = \nu_i^j(q,\sigma) \tag{3.9}$$

$$\omega_i^j(q,\sigma)! \Rightarrow \alpha_i^j(q',\sigma) = \tilde{q}' \text{ for } \omega_i^j(q,\sigma) = \tilde{q} \tag{3.10}$$

For each $q \in Q_i^j$ and $\sigma \in \Sigma_{\text{req}} \setminus \{j_{\text{req}}\}$

$$\alpha_i^j(q,\sigma) = q' \tag{3.11}$$

For each $\sigma \in \Sigma^{\text{rec}} \setminus \{j_{\text{st}}\}$

$$\alpha_i^j(\text{W},\sigma) = \text{W} \tag{3.12}$$

$$\alpha_i^j(\text{W}, j_{\text{st}}) = q_{0,i}^j \text{ and } \alpha_i^j(q_{0,i}^{j\prime}, j_{\text{fin}}) = \text{W} \tag{3.13}$$

This means, that the request event can happen from each state of the supervisor states Q_i^j, leading to the corresponding state in the supervisor for state attraction T_i^j. After the active configuration is finished, the finish event j_{fin} occurs and leads the reconfiguration supervisor R_i^j to the waiting state W. From there, a transition with the start event j_{st} starts the configuration from the initial state of the supervisor S_i^j.

For our example, we have the automata R_1^{conf1} and R_1^{conf2} for the two configurations $conf1$ and $conf2$. These reconfiguration supervisors are big automatons to shown here.

Reconfiguration Coordinator Automata

In order to enforce the change between configurations, we introduce the coordination automaton $P^j = (V^j, \Sigma^{\text{rec}}, \xi^j, v_0^j, V_{\text{m}}^j)$ for each $j \in \mathcal{C}$. This automaton performs the sequence of events $j_{req} \rightarrow j_{fin} \rightarrow j_{st}$ that are necessary in a reconfiguration. The automaton P^j for configuration $j \in \mathcal{C}$ is constructed with the following rules:

$$V^j = \{1,2,3\} \text{ and } v_0^j = 1 \text{ and } V_{\text{m}}^j = \{1\} \tag{3.14}$$

$$\xi^j(1, j_{\text{req}}) = 2 \text{ and } \xi^j(1,\sigma) = 1 \text{ for all } \sigma \in \Sigma^{\text{rec}} \setminus \{1_{\text{req}}\} \tag{3.15}$$

$$\xi^j(2, k_{\text{fin}}) = 3, k \in \mathcal{C} \setminus \{j\} \text{ and } \xi^j(2,\sigma) = 2, \sigma \in \Sigma^{\text{rec}} \setminus \Sigma_{\text{fin}} \tag{3.16}$$

$$\xi^j(3, j_{\text{st}}) = 1 \tag{3.17}$$

In words, this automaton is constructed with three states, and the transitions between these state are

- from state 1 to state 2 there is a transition with the request event j_{req} and there are selfloop transitions with all other events in state 1. That is, P_i^j is able to follow any configuration in state 1 until a request for configuration j happens.
- from state 2 to state 3, there is a transition with the finish event $k_{fin} \in \Sigma_{fin} \setminus \{j_{fin}\}$ and there are selfloops with all events in state 2. That is, P_i^j is able to wait in state 2 until any other configuration k is completed and then execute the finish event k_{fin}
- from state 3, there is only a transition with the start event j_{st}. That is, configuration j starts after the previous configuration is finished.

As an example, the coordination automaton P^{conf1} for configuration 1 of our example system is shown in Figure 3.20.

3.2.3 Reconfiguration Control for Module 2

We use the plant model of module 2 as we introduced before. The realization of the desired behavior for each configuration, can be done by following a safety specification and computing the relevant configuration supervisors. Here, we introduce the specifications for this module and for each configuration. We consider C_2^1 for configuration (1) and C_2^2 for configuration (2).

In configuration 1, the product enters from (MA2) to (RT2) in the second module, is processed in MA3 and then moves back to (RT2) and delivered to the outside. For configuration 2, C_2^2 is an empty automaton because module 2 does not participate in configuration 2. We can see all the specification automata in Figure 3.21 and Figure 3.22. Here, S_2^1 and S_2^2 for module 2 can be determined using the *SupCon* algorithm for the plant and the related specifications of our configurations. The automaton of S_2^2 is depicted in Figure 3.23, and satisfies that $L_{\mathrm{m}}(S_2^1) = SupC(L_{\mathrm{m}}(C_2^1), G_{\mathrm{RMS2}}, \Sigma_{\mathrm{u}})$ and $L_{\mathrm{m}}(S_2^2) = SupC(L_{\mathrm{m}}(C_2^2), G_{\mathrm{RMS2}}, \Sigma_{\mathrm{u}})$.

Now, the computation of the reconfiguration supervisor for module 2 is performed using the method in section 3.2.2. The statistics for the resulting supervisors are

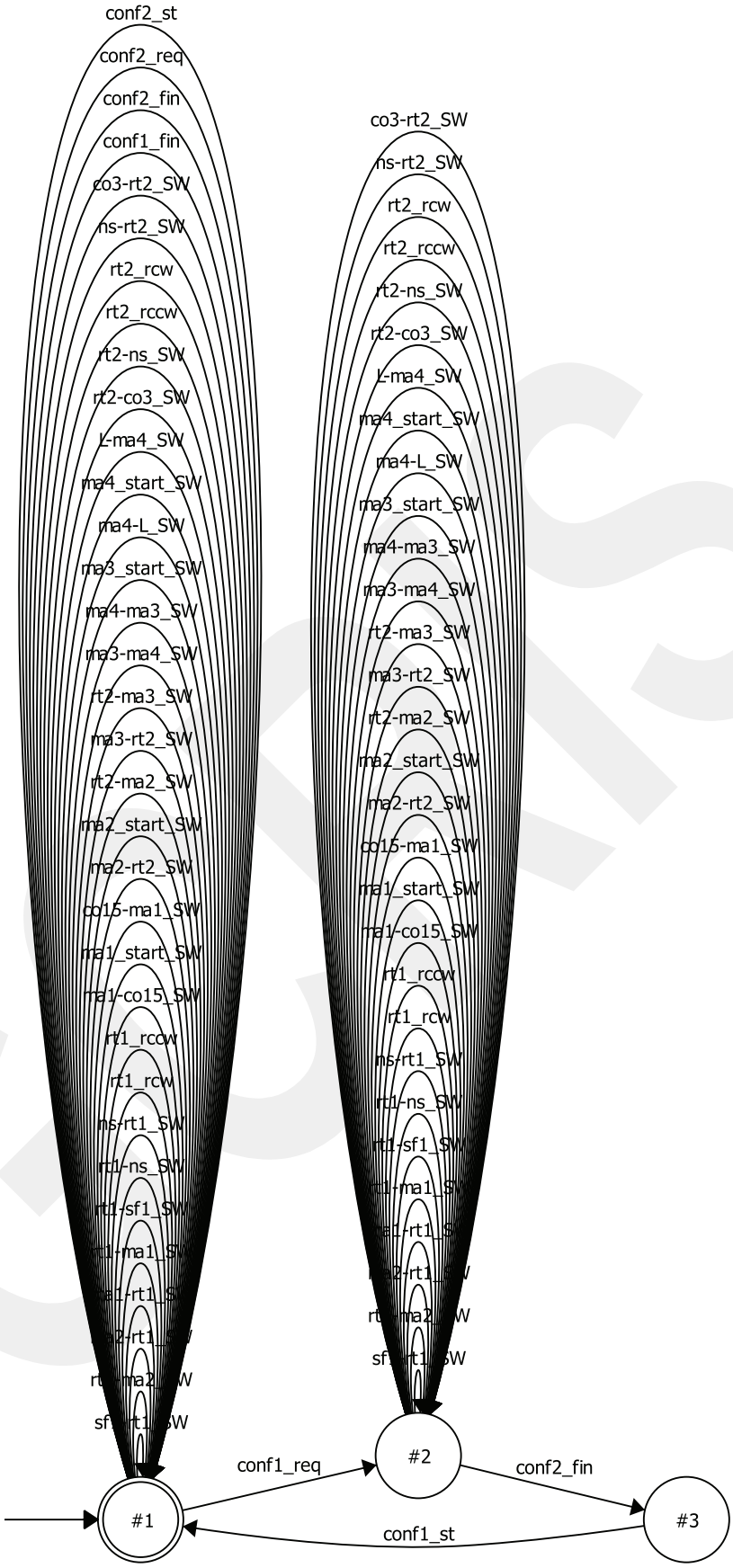

Figure 3.20: Automatoa of P^{conf1}

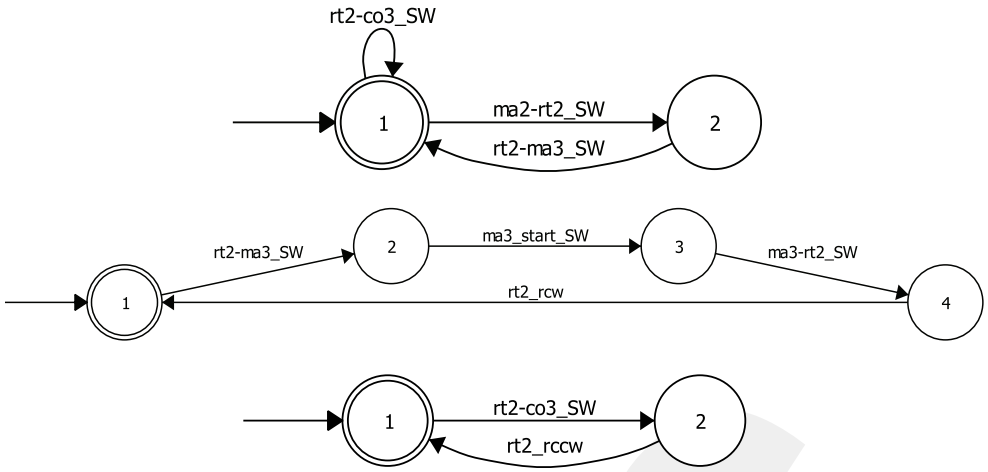

Figure 3.21: Specification of configuration 1

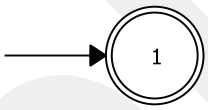

Figure 3.22: Specification of configuration 2

given in Table 3.5. The automata for the reconfiguration supervisors, are shown in Figure 3.24 and Figure 3.25

Table 3.5: Module 2 supervisors

SUPERVISORS	STATES	EVENTS	TRANSITIONS
$supervisor_{\text{con1}}$	7	16	7
$supervisor_{\text{con2}}$	11	6	0
R_{con1}	15	22	44
R_{con2}	3	22	24

3.3 RECONFIGURATION CONTROL WITH HIERARCHICAL ABSTRACTION

3.3.1 Description of Abstraction-Based Control

In the previous section, the computation of modular reconfiguration supervisors for each RMS module is described. We next address the computation of abstraction-based supervisors that coordinate the operation of the different mod-

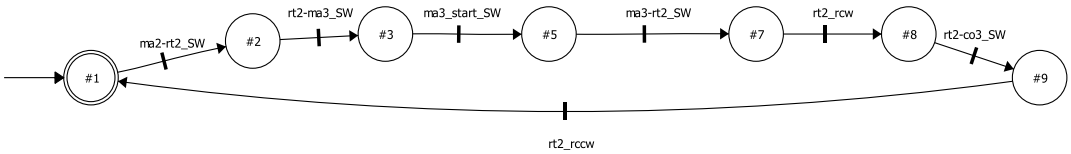

Figure 3.23: Module 2 supervisors

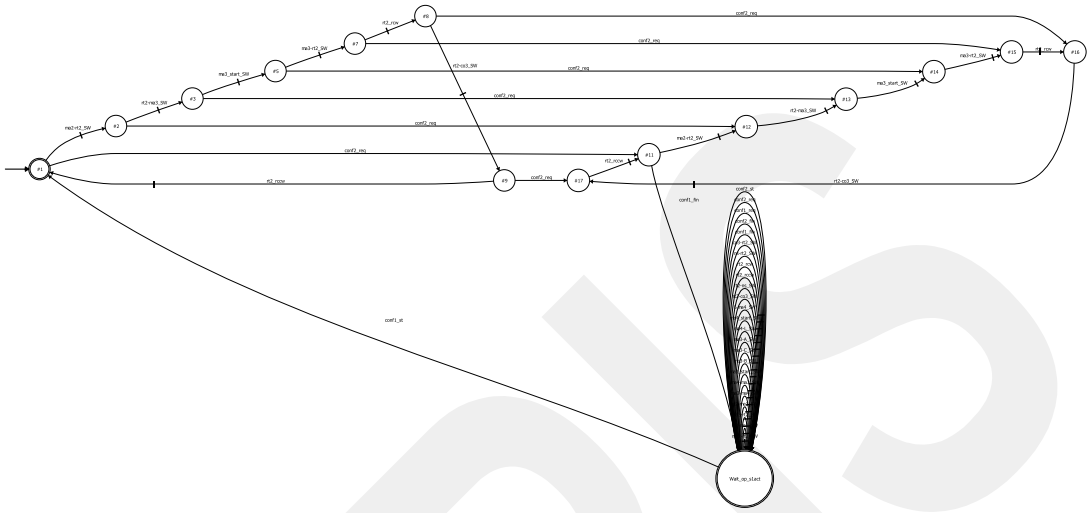

Figure 3.24: Module 2 R_{con1}

ules. The basis for this computation are the previously determined supervisors for the first module S_1^1 and S_1^2 and for the second module S_2^1 and S_2^2.

Using the method in section 1.6, we first compute an abstraction-based supervisor $\hat{S}^1$ (low-level models S_1^1 and S_2^1) for configuration 1 and $\hat{S}^2$ (low-level models S_1^2 and S_2^2) for configuration 2. The resulting automata are shown in Figure 3.26.

As the next step, we compute the reconfiguration supervisors for the abstraction-based supervisors. To this end, we need state attraction supervisors in order to complete each active configuration. The state attraction supervisors for configuration 1 and configuration 2 are shown in Figure 3.27.

However, we now note that the abstraction-based approach requires a modification of the state attraction supervisors. The reason is that state attraction only needs to be achieved in the high level, whereas the low-level supervisors will follow the operation of the high level-supervisor. That is, we propose to use a modified state attraction supervisor in the low level, whereby all transitions with events that are shared with the high-level supervisor are preserved when computing the supervisor for state attraction. This idea is illustrated comparing the supervisors

Figure 3.25: Module 2 R_{con2}

for state attraction in Figure 3.27 and 3.28. Although the transitions with event `sf-rt1_SW` in Figure 3.28 are removed when computing the state attraction supervisor in Figure 3.28, there are preserved in the modified state attraction supervisor since `sf-rt1_SW` is considered as a shared event. The same idea is applied to all transitions that appear in Figure 3.28 but not in Figure 3.27.

Finally, the automata for the computed high level reconfiguration supervisors are in Figure 3.29. Note that these reconfiguration supervisors are constructed from the supervisors automata and the modified state attraction supervisors.

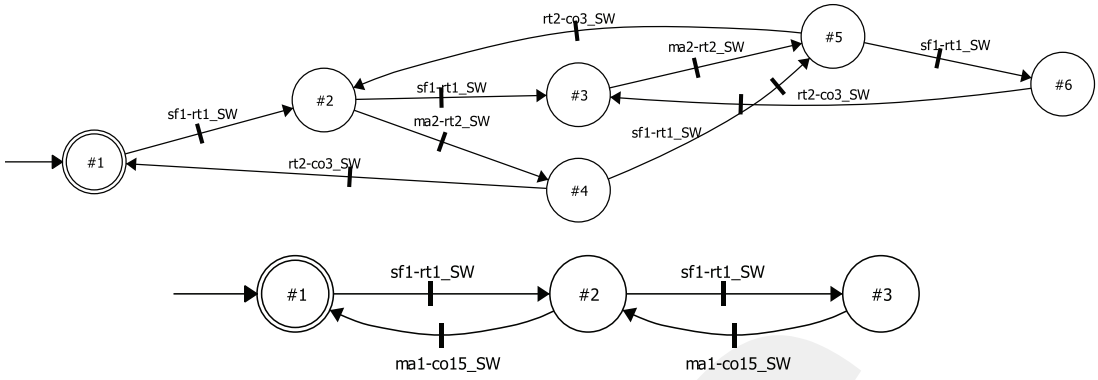

Figure 3.26: High level supervisors

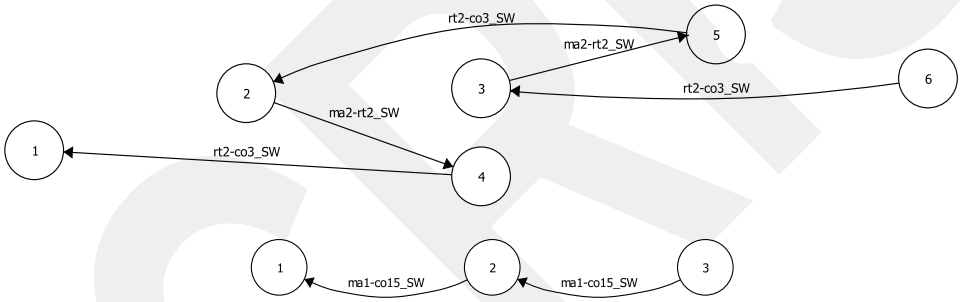

Figure 3.27: State attraction supervisors for the high-level supervisors

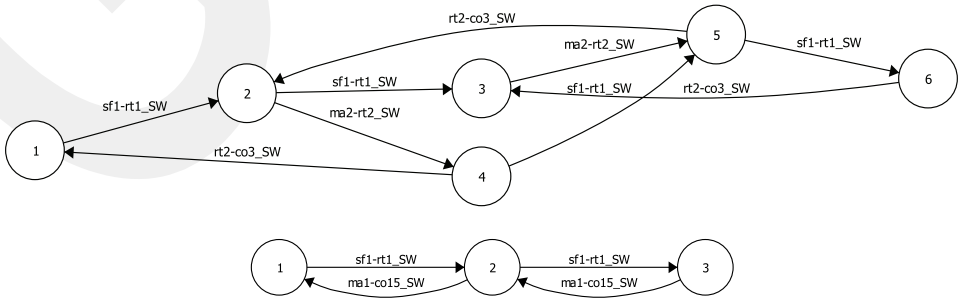

Figure 3.28: Modified state attraction supervisors

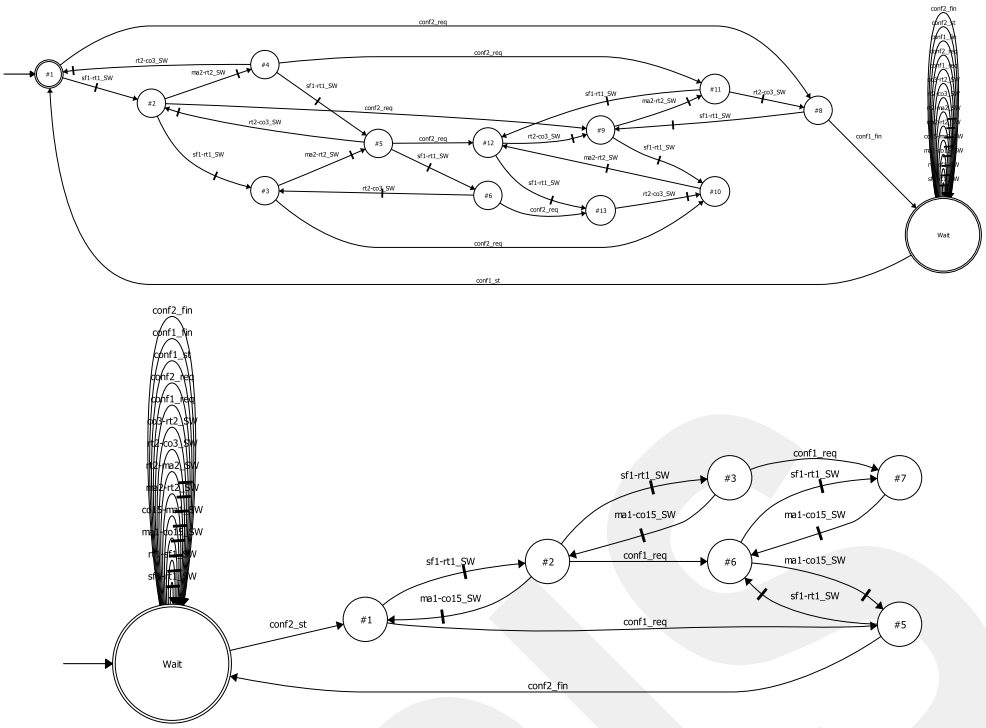

Figure 3.29: High level reconfiguration supervisor 1 and 2

3.3.2 Formalization of the Reconfiguration Problem

We use the same notation as before. In addition, we assume that the realization of any configuration $j \in \mathcal{C}$ can be done by abstraction-based supervisors, such that $S_i^j = (Q_i^j, \Sigma_i, \nu_i^j, q_{0,i}^j, Q_{\mathrm{m},i}^j)$ are the component supervisors as in Section 3.2.2 and $\hat{S}^j = (\hat{Q}^j, \hat{\Sigma}^j, \hat{\nu}^j, q_0^j, Q_{\mathrm{m}}^j)$ for the high-level supervisors for $i = 1, \ldots, n$. In order to achieve a nonblocking supervisors for each configuration $j \in \mathcal{C}$, presented as $(||_{i=1}^n S_i^j)||\hat{S}^j||G \subseteq (||_{i=1}^n K_i^j)||\hat{K}^j$, we suppose that the natural projections $p_i^j : \Sigma_i^\star \to (\Sigma_i \cap \hat{\Sigma}^j)^\star$ are natural observers.

We compute a modular reconfiguration supervisors $R_i^j = (Z_i^j, \Sigma_i^{\mathrm{rec}}, \alpha_i^j, z_{0,i}^j, Z_{\mathrm{m},i}^j)$ as described in Section 3.2.2 with the modification that the transitions with shared events are added in the supervisors for state attraction T_i^j. In addition, we compute the abstraction-based modular reconfiguration supervisor $\hat{R}^j = (\hat{Z}^j, \hat{\Sigma}^{j,\mathrm{rec}}, \hat{\alpha}^j, \hat{z}_0^j, \hat{Z}_{\mathrm{m}}^j)$ for each configuration $j \in \mathcal{C}$ using the computation in Section 3.2.2.

The overall reconfiguration supervisor is then given as the synchronous composi-

tion of all the modular reconfiguration supervisors

$$Supervisor^{\text{rec}} = ||_{j\in\mathcal{C}}\big((||_{i=1}^{n} R_i^j)||\hat{R}^j||P^j\big). \tag{3.18}$$

It has to be noted that all supervisors are computed with polynomial complexity.

3.3.3 Overall Example System

We summarize the previously obtained results for configuration 1 and 2 of the example RMS. The computed modular reconfiguration supervisors and the high level supervisors are denoted as R_1^1, R_1^2, R_2^1, R_2^2, and $\hat{R}^1$, $\hat{R}^2$. The related statistics are listed in Table 3.6.

Table 3.6: Reconfiguration supervisors

SUPERVISORS	STATES	EVENTS	TRANSITIONS
R_1^1	43	22	110
R_1^2	39	22	100
R_2^1	15	25	47
R_2^2	3	25	27
$\hat{R}^1$	13	14	35
$\hat{R}^2$	7	14	24

We also note that the coordination automata P^1 and P^2 were computed before in section 3.2.2.

3.4 SUMMARY

The abstraction-based reconfiguration control is useful to deal with large-scale RMS with many components. Our new method allows the design of reconfiguration supervisors such that

- the active configuration is completed before starting a new configuration,
- a new configuration can be requested at any time

- the new configuration becomes active after a bounded number of event occurrences

We further note an important fact of our design. The modular reconfiguration supervisors on the low level and on the high level have the same structure. As a result, it is possible to apply the described method not only for hierarchies with two levels but for hierarchies with an arbitrary number of levels. An example of reconfiguration control with 4 levels is given in the next chapter.

CHAPTER IV

APPLICATION EXAMPLE OF OVERALL RMS

In this chapter, we apply the method we presented in chapter II and III to a part of a large-scale RMS example that is available at the Department of Mechatronics Engineering, Çankaya University. The picture of the RMS components is depicted in Figure 4.1. This application example of an RMS, consists of 18 different components, such that it is necessary to divide the RMS into several modules and apply the abstraction-based design in order to reduce the design complexity.

This RMS example consist of

- One stack feeder: SF1,
- One exit slide: RC,
- Four rotary tables: RT1, RT2, RT6, and RT7,
- Three conveyor belts: CO3, CO15, and CO16,
- Two rail transport systems: RTS1 and RTS2,
- Four single production machines: MA1, MA4, MA9 and MA11,
- Three RMTs: MA2, MA3 and MA10.

Before introducing the RMS modules, we first present the models of the RMS components in the form of automata. Hereby, the models of the rotary tables, conveyor belts, stack feeder single production machines and RMTs are analogous to the models introduced in chapter II and III. Hence, we focus on the models of the components that were not considered up to now.

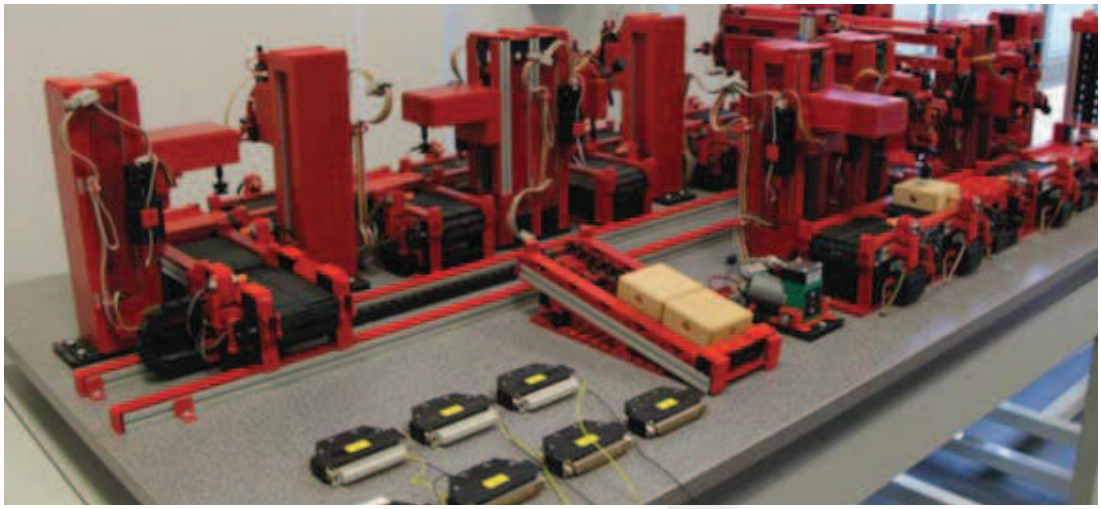

Figure 4.1: Picture of overall RMS

4.1 MODELS OF NEW RMS COMPONENTS

4.1.1 Rail Transport Systems

A picture of this component of the RMS is shown in Figure 4.2. The rail transport

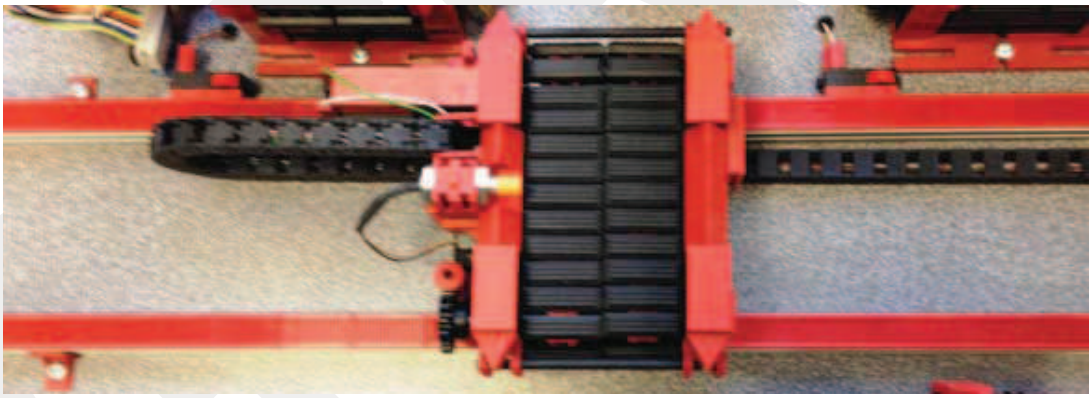

Figure 4.2: Picture of a rail transport system.

system (RTS) consists of two main parts. A cart that moves on a rail and a conveyor belt. The cart can stop at different positions, whereas the conveyor belt can transport products to the upper and lower direction. In the RMS, there are two different RTS components that are located on the same rail as can be seen in Figure 4.3.

In the following, we model each RTS component by a finite state automaton that is composed of one model for the cart and one model of the conveyor belt.

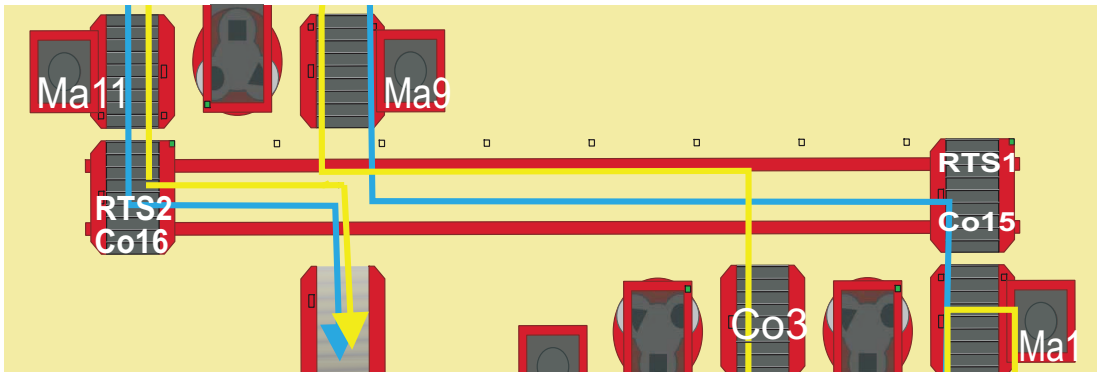

Figure 4.3: RTS components RTS1 and RTS2.

Model Of RTS1

The operation of RTS1 is realized by the composition of the conveyor belt CO15 and the movement of the RTS denoted by MOV1. There are four fixed positions where RTS1 can stop which are denoted as position 5 to position 2. Position 5 is the initial position of RTS1 on the right-hand side. In each position there are multiple neighbor components, where the conveyor belt CO15 can move the products between the RTS1 and its neighbors. Due to this, the RTS1 component can move and transport products to different places. The plant automaton of CO15 is denoted as G_{CO15} and it has 20 states, 25 events, and 37 transitions. The movement automaton of RTS1 has 28 states, 21 events, and 41 transitions and is denoted as G_{MOV1}.

Moreover, we recall the concept of the abstraction-based supervisory control and abstract these models such that we obtain small automata for each of our two components. We write $G^{\text{high}}_{\text{CO15}}$ and $G^{\text{high}}_{\text{MOV1}}$ for the abstracted models and the overall model of RTS1 is

$$G_{\text{RTS1}} = G^{\text{high}}_{\text{CO15}} || G^{\text{high}}_{\text{MOV1}}. \tag{4.1}$$

The abstracted automata for each CO15 and MOV1 are shown in Figure 4.4 and Figure 4.5. All related events for the RTS1 movement are listed in Table 4.1 and for events for CO15 are shown in Table 4.2.

Finally, the overall plant result automata for the rail transport system RTS1 has 40 states, 22 events, and 222 transitions.

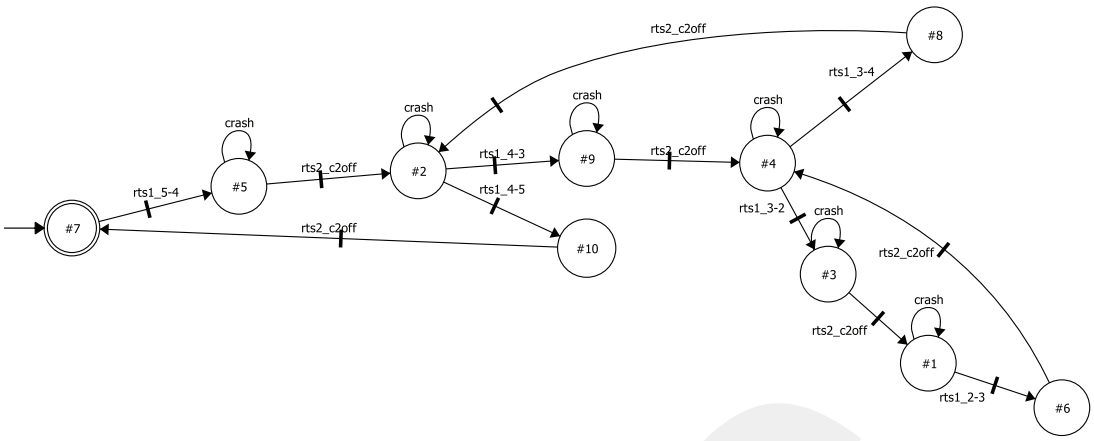

Figure 4.4: Abstracted RTS1

Table 4.1: Rail transport system events RTS1

EVENTS NAMES	DESCRIPTION	STATUS
rts1_5-4	RTS1 can move from position 5 to 4	C
rts1_4-5	RTS1 can move from position 4 to 5	C
rts1_4-3	RTS1 can move from position 4 to 3	C
rts1_3-4	RTS1 can move from position 3 to 4	C
rts1_3-2	RTS1 can move from position 3 to 2	C
rts1_2-3	RTS1 can move from position 2 to 3	C
rts2_c2off	RTS1 stop (motor switch off)	C
crash	If RTS1 and RTS2 arrives in same position	unC

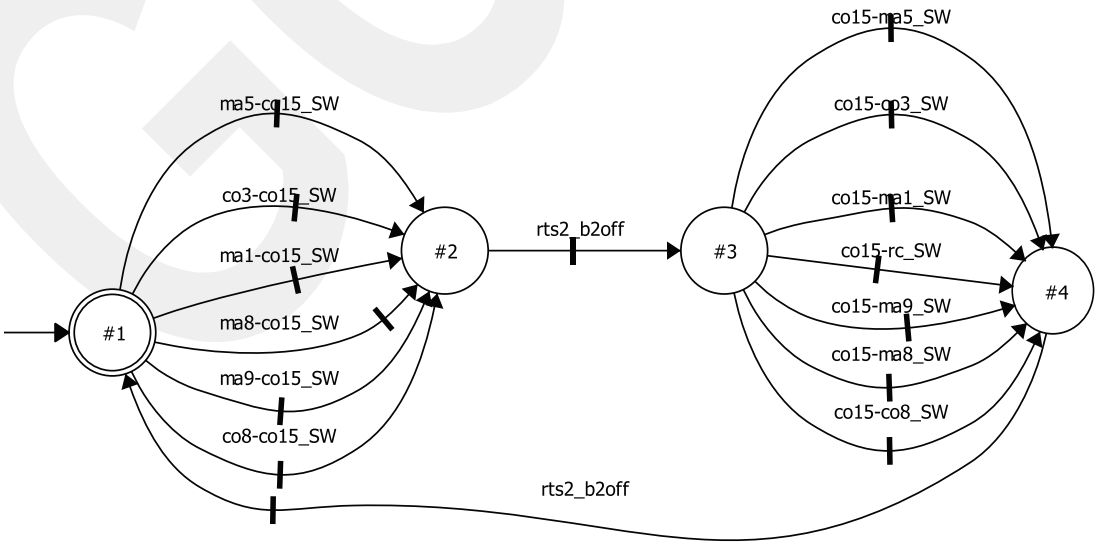

Figure 4.5: Abstracted CO15

Table 4.2: Conveyor belt events CO15

EVENTS NAMES	DESCRIPTION	STATUS
co15-ma1_SW	move products from (CO15) to (MA1)	C
ma1-co15_SW	move products from (MA1) to (CO15)	C
co15-ma5_SW	move products from (CO15) to (MA5)	C
ma5-co15_SW	move products from (MA5) to (CO15)	C
co15-co3_SW	move products from (CO15) to (CO3)	C
co3-co15_SW	move products from (CO3) to (CO15)	C
co15-co8_SW	move products from (CO15) to (CO8)	C
co8-co15_SW	move products from (CO8) to (CO15)	C
co15-ma8_SW	move products from (CO15) to (MA8)	C
ma8-co15_SW	move products from (MA8) to (CO15)	C
co15-ma9_SW	move products from (CO15) to (MA9)	C
ma9-co15_SW	move products from (MA9) to (CO15)	C
co15-rc_SW	move products from (CO15) to (RC)	C
rts2_b2off	conveyor belt stop (motor switch off)	C

Model Of RTS2

The plant model of the rail transport system RTS2 is analogous to the plant model of RTS1. The only difference is that the initial position of RTS2 is position 1, and RTS2 only can stop as positions 1 to 4. The overall plant automaton G_{RTS2} has has 40 states, 20 events, and 202 transitions and is composed of G_{CO16} with 18 states, 22 events, and 32 transitions and G_{MOV2} with 28 states, 20 events, and 45 transitions.

From the abstraction-based supervisory control we have the abstracted automata for $G^{\text{high}}_{\text{CO16}}$ and $G^{\text{high}}_{\text{MOV2}}$ which are depicted in Figure 4.6 and Figure 4.7. The related events for the RTS2 movement and for CO16 are listed in Table 4.3 and Table 4.4, respectively. That is

$$G_{\text{RTS2}} = G^{\text{high}}_{\text{CO16}} || G^{\text{high}}_{\text{MOV2}} \tag{4.2}$$

4.1.2 Model Of The Exit Slide

The processed products can leave the RMS through the exit slide. The arrival of a products inside the exit slide is indicated by a hardware sensor. In fact,

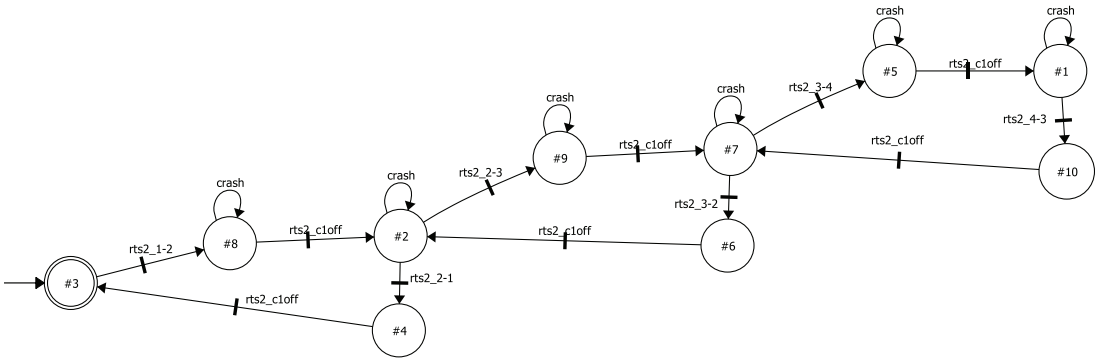

Figure 4.6: Abstracted RTS2

Table 4.3: Rail transport system events RTS2

EVENTS NAMES	DESCRIPTION	STATUS
rts2_1-2	RTS2 can move from position 1 to 2	C
rts2_2-1	RTS2 can move from position 2 to 1	C
rts2_2-3	RTS2 can move from position 2 to 3	C
rts2_3-2	RTS2 can move from position 3 to 2	C
rts2_3-4	RTS2 can move from position 3 to 4	C
rts2_4-3	RTS2 can move from position 4 to 3	C
rts2_c1off	RTS2 stop (motor switch off)	C
crash	If RTS2 and RTS1 arrives in same position	unC

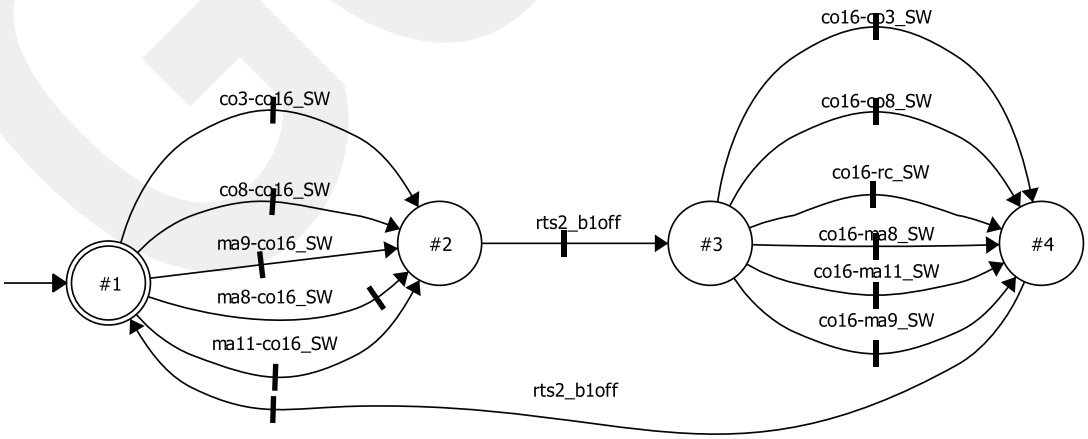

Figure 4.7: Abstracted CO16.

Table 4.4: Conveyor belt events CO16

EVENTS NAMES	DESCRIPTION	STATUS
co16-ma11_SW	move products from (CO16) to (MA11)	C
ma11-co16_SW	move products from (MA11) to (CO16)	C
co16-ma9_SW	move products from (CO16) to (MA9)	C
ma9-co16_SW	move products from (MA9) to (CO16)	C
co16-ma8_SW	move products from (CO16) to (MA8)	C
ma8-co16_SW	move products from (MA8) to (CO16)	C
co16-co3_SW	move products from (CO16) to (CO3)	C
co3-co16_SW	move products from (CO3) to (CO16)	C
co16-co8_SW	move products from (CO16) to (CO8)	C
co8-co16_SW	move products from (CO8) to (CO16)	C
co16-rc_SW	move products from (CO16) to (RC)	C
rts2_b1off	conveyor belt stop (motor switch off)	C

the component exit slide is represented as the storage or the deposit area for the products after processing in the system. The picture of the exit slide is shown in Figure 4.8. In our RMS, products can leave the system from RTS1 or RTS2 to the exit slide RC with the events (co16-rc_SW and co15-rc_SW). The automaton for modeling the exit slide and its high-level abstraction are shown in Figure 4.9. The related events are illustrated in Table 4.5.

Table 4.5: Exit slide events

EVENTS NAMES	DESCRIPTION	STATUS
co16-rc_SW	product move from (CO16) to (RC)	C
co15-rc_SW	product move from (CO15) to (RC)	C
rc_wpar	sensor detects products arrives	unC
rc_wplv	sensor detects products leaves	unC

4.2 ABSTRACTION-BASED RECONFIGURATION SUPERVISORS OF OVERALL RMS

In this section, the method of abstraction-based reconfiguration control is applied on the large-scale RMS example. We introduce the the models of plants, specifications, configuration supervisors, and reconfiguration supervisors for the large-scale RMS example. Figure 4.11 shows the overview of the system.

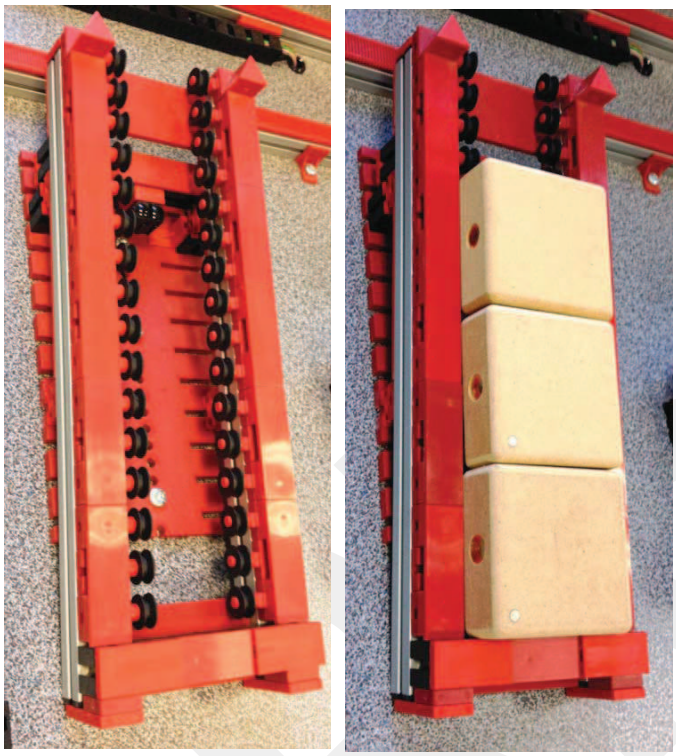

Figure 4.8: Picture of the exit slide

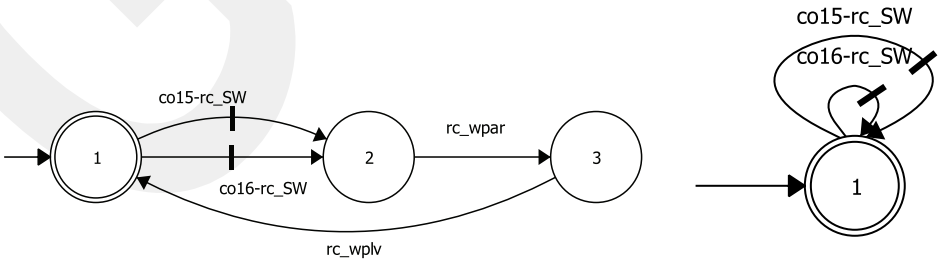

Figure 4.9: Automata models of the exit slide.

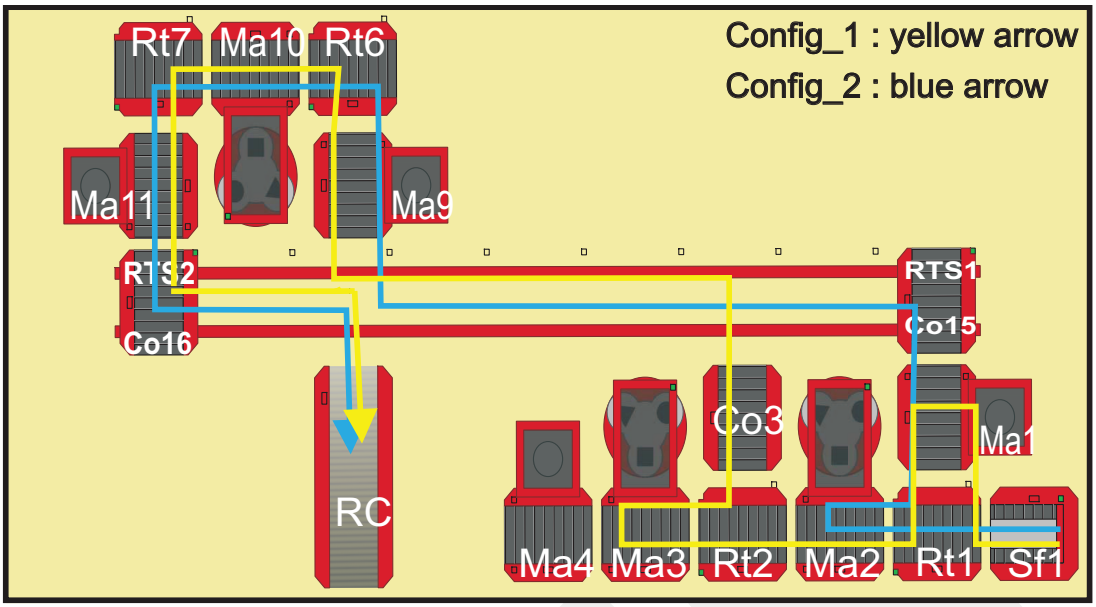

Figure 4.10: RMS overview.

We consider two main configurations for our RMS. These configurations are given by the product paths in Figure 4.10 and are denoted as *configuration 1 and configuration 2*:

- Configuration(1) : *in this configuration, products can enter the system from SF1, move to RT1 and then to MA1. After processing in MA1 products move back to RT1 and leave to processing in MA2 with the first machine tool. From MA2, they move to RT2 and to MA3 where processing happens and they move again to RT2. Then, products move to CO3. When the products are in CO3, the component RTS2 moves from its initial position and stops at position 3. RTS2 takes the products from CO3 and transports them to MA9 in position 2. Products are processed by MA9 and move to RT6. Furthermore, products leave to RT6 and from there to MA10. After processing, products finally move through RT7 and MA11 to RTS2 again and are deposited in the exit slide RC.*

- Configuration(2): *In this configuration, products also enter from SF1. After moving to RT1, they move to MA2 and are processed by the second machine tool of this RMT. Then, products move back to RT1 and are forwarded for processing in MA1. The component RTS1 then transport products from MA1 in position 5 to MA9 in position 2. There is no processing in MA9, and products directly move through RT6, MA10, and RT7 to MA11. After processing in MA11, RTS2 transports products to RC.*

Since the RMS has a large number of components we next divide the RMS into 8 different modules. These modules are shown in Figure 4.11. The description of the modules is given as follows:

- Module 1 consists of SF1, RT1, MA1, and MA2,
- Module 2 consists of RT2, MA3, and MA4,
- Module 3 consists of CO3,
- Module 4 consists of RTS1, and CO15,
- Module 5 consists of RTS2, and CO16,
- Module 6 consists of MA9,
- Module 7 consists of RT6, MA10, RT7, and MA11,
- Module 8 consists of RC.

Note that the modules are chosen such that always neighboring components that form functional entities are put into the same module.

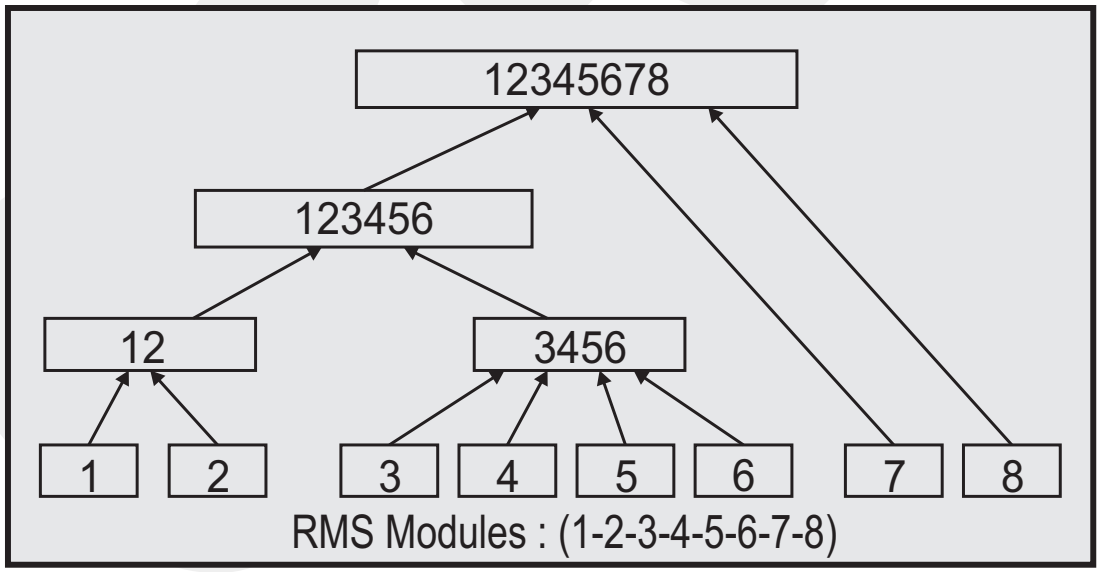

Figure 4.11: Overview of RMS hierarchical modules

We next perform an abstraction-based design of the reconfiguration supervisor for our RMS in four different levels. The basic strategy of the abstraction-based design is shown in Figure 4.11 and Figure 4.12. The computation for the different modules and levels is performed in the sequel.

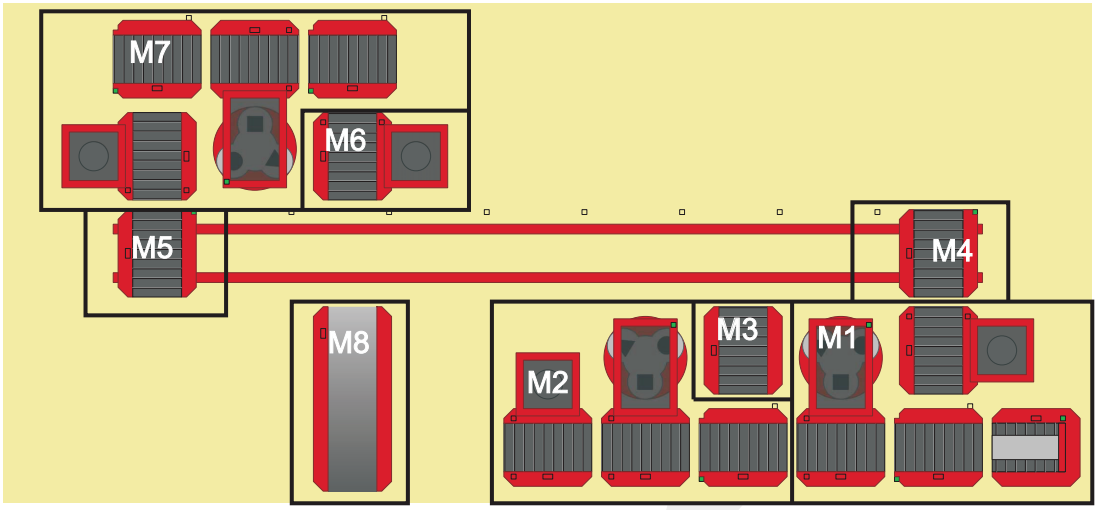

Figure 4.12: Overview of RMS modules

4.3 RMS MODULE 1

The supervisor design for this module is already described in section 3.2.1. The structure of the design is shown in Figure 4.13. The plant for this module can be written as

$$G_1 = G^{\text{high}}_{\text{SF1}} || G^{\text{high}}_{\text{RT1}} || G^{\text{high}}_{\text{MA1}} || G^{\text{high}}_{\text{MA2}} \tag{4.3}$$

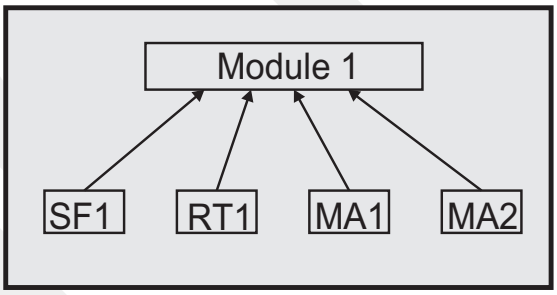

Figure 4.13: RMS module 1

The overall plant automaton G_1 is too big to be shown here. It has 16 states, 16 events, and 88 transitions. In addition, we use the same specifications as before for computing the configuration supervisors. The statistics of all supervisors and the abstraction supervisors are summarized in Table 4.6.

Table 4.6: Module 1 reconfiguration supervisors

SUPERVISORS	STATES	EVENTS	TRANSITIONS
S_1^1	21	16	33
S_1^2	19	16	29
$\hat{S}_1^1$	3	5	4
$\hat{S}_1^2$	3	5	4
R_1^1	43	22	113
R_1^2	39	22	103

4.4 RMS MODULE 2

The reconfiguration supervisor design for this module is analogous to the description in section 3.3. The overall structure of this module is shown in Figure 4.14. The overall plant automaton has 16 states, 16 events, and 84 transitions. It is denoted as

$$G_2 = G_{\text{RT2}}^{\text{high}} || G_{\text{MA3}}^{\text{high}} || G_{\text{MA4}}^{\text{high}} \tag{4.4}$$

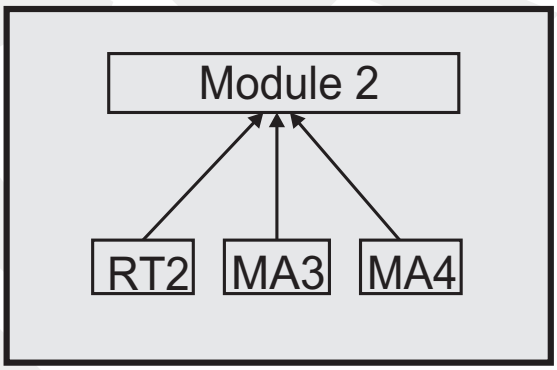

Figure 4.14: RMS module 2

The computation for the configuration supervisors and the reconfiguration supervisors are as in section 3.3 and the resulting statistics are shown in Table 4.7.

4.5 RMS MODULE 3

This module only consists of one conveyor belt as in Figure 4.15.

Table 4.7: Module 2 reconfiguration supervisors

SUPERVISORS	STATES	EVENTS	TRANSITIONS
S_2^1	7	16	7
S_2^2	1	16	0
$\hat{S}_2^1$	2	4	2
$\hat{S}_2^2$	1	4	0
R_2^1	15	25	47
R_2^2	3	25	27

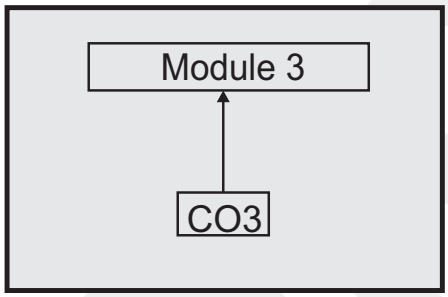

Figure 4.15: RMS module 3

The model for the conveyor belt CO3 is realized according to its neighbors. The lower neighbor component is RT2, and from the upper side the products can arrive to CO3 or leave to RTS1 or RTS2 with the events(`co15-co3_SW`, `co3-co15_SW`) for RTS1, and (`co16-co3_SW`, `co3-co16_SW`) for RTS2. The low level model of CO3 is shown in Figure 4.16, The abstracted automata that can be the plant model of CO3 is written as $G_{\text{CO3}}^{\text{high}}$ and it is depicted in Figure 4.17.

$$G_3 = G_{\text{CO3}}^{\text{high}} \tag{4.5}$$

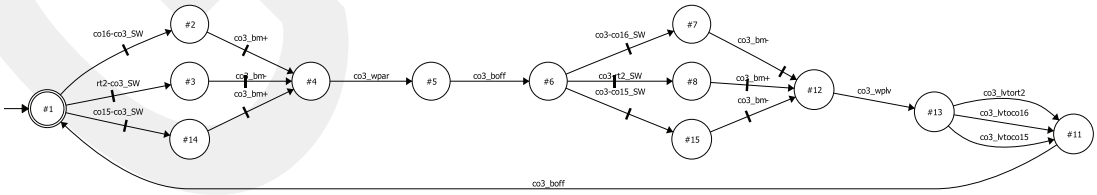

Figure 4.16: Low level model Of CO3

In the first configuration of module 3, products can move from RT2 to CO3 and after that should leave to CO16 on RTS2. For the second configuration module 3 is not used. The associated specifications are shown in Figure 4.18 and 4.19.

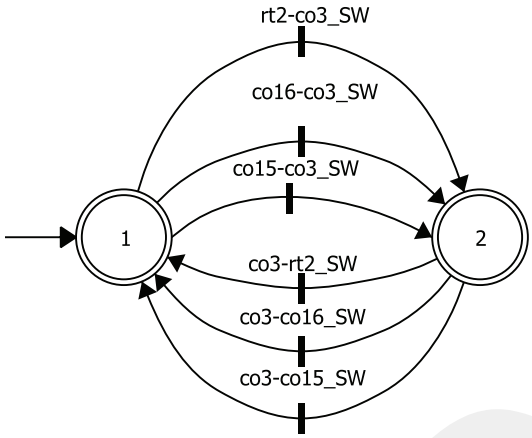

Figure 4.17: High level model Of CO3

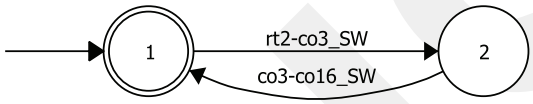

Figure 4.18: Module 3 configuration 1 specifications

Finally, by using these specifications, the configuration supervisors are computed and Table 4.8 shows these results.

Table 4.8: Module 3 reconfiguration supervisors

SUPERVISORS	STATES	EVENTS	TRANSITIONS
S_3^1	2	6	2
S_3^2	1	6	0
$\hat{S}_3^1$	2	6	2
$\hat{S}_3^2$	1	6	0
R_3^1	5	12	19
R_3^2	3	12	14

4.6 RMS MODULE 4

The plant for module 4 is realized by the rail transport system RTS1 and CO15, such that the overall plant is composed of the high-level abstraction of RTS1 and the high-level abstraction of CO15, as shown in Figure 4.20. That is, using the

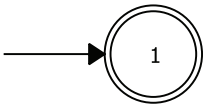

Figure 4.19: Module 3 configuration 2 specifications

result from section 4.1, we compute

$$G_4 = G^{\text{high}}_{\text{RTS1}} || G^{\text{high}}_{\text{CO15}} \tag{4.6}$$

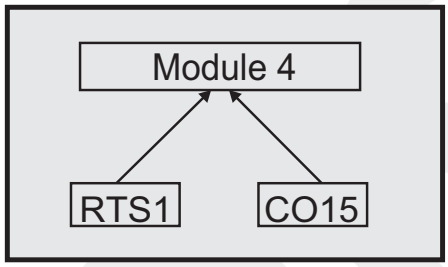

Figure 4.20: RMS module 4

The overall plant automata has 40 states, 22 events, and 222 transitions. The desired operation of module 4 is described as follows.

- *In configuration 1, RTS1 and CO15 should never operate. Due to this, all events are disabled.* The corresponding specification automaton C_4^1 is shown in Figure 4.21.

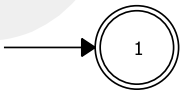

Figure 4.21: Module 4 configuration 1 specifications

- *In configuration 2, products should move from MA1 to CO15 only if RTS1 in its initial position (position 5).* The corresponding specification automaton $C_4^{2,1}$ is shown in Figure 4.22 (a).

- *In configuration 2, RTS1 moves from position 5 to position 4, position 3, and finally to position 2. Then products are delivered by CO15 to Ma9.* The corresponding specification automaton $C_4^{2,2}$ is shown in Figure 4.22 (b).

- *In configuration 2, if a product was moved from CO15 to MA9, then RTS1 should move back to position 5 through position 2,3, and 4.* The corresponding specification automaton $C_4^{2,3}$ is shown in Figure 4.22 (c).

- *If the rail transport system RTS1 is moving, then the conveyor belt CO15 should not move.* The corresponding specification automaton $C_4^{2,4}$ is shown in Figure 4.22 (d).

- *If the conveyor belt CO15 is moving products, then the rail transport system RTS1 should not move.* The corresponding specification automaton $C_4^{2,5}$ is shown in Figure 4.22 (e).

Using the described specifications, the configuration supervisors and reconfiguration supervisors are computed. The resulting sizes are listed in Table 4.9.

Table 4.9: Module 4 reconfiguration Supervisors

SUPERVISORS	STATES	EVENTS	TRANSITIONS
S_4^1	1	22	0
S_4^2	16	22	26
$\hat{S}_4^1$	1	19	0
$\hat{S}_4^2$	8	19	8
R_4^1	3	28	30
R_4^2	33	28	87

4.7 RMS MODULE 5

RTS2 and CO16 are the components of this module as described in section 4.1. The structure of the module is shown in Figure 4.23. The plant automaton is computed as

$$G_5 = G_{\text{RTS2}}^{\text{high}} || G_{\text{CO16}}^{\text{high}} \tag{4.7}$$

and it has 40 states, 20 events, and 202 transitions. In this module, the first configuration has to be obtained for two different paths. In path 1, the rail transport moves products from CO3 to processing in MA9, and in path 2 the rail transport system moves products from MA11 to the exit slide RC.

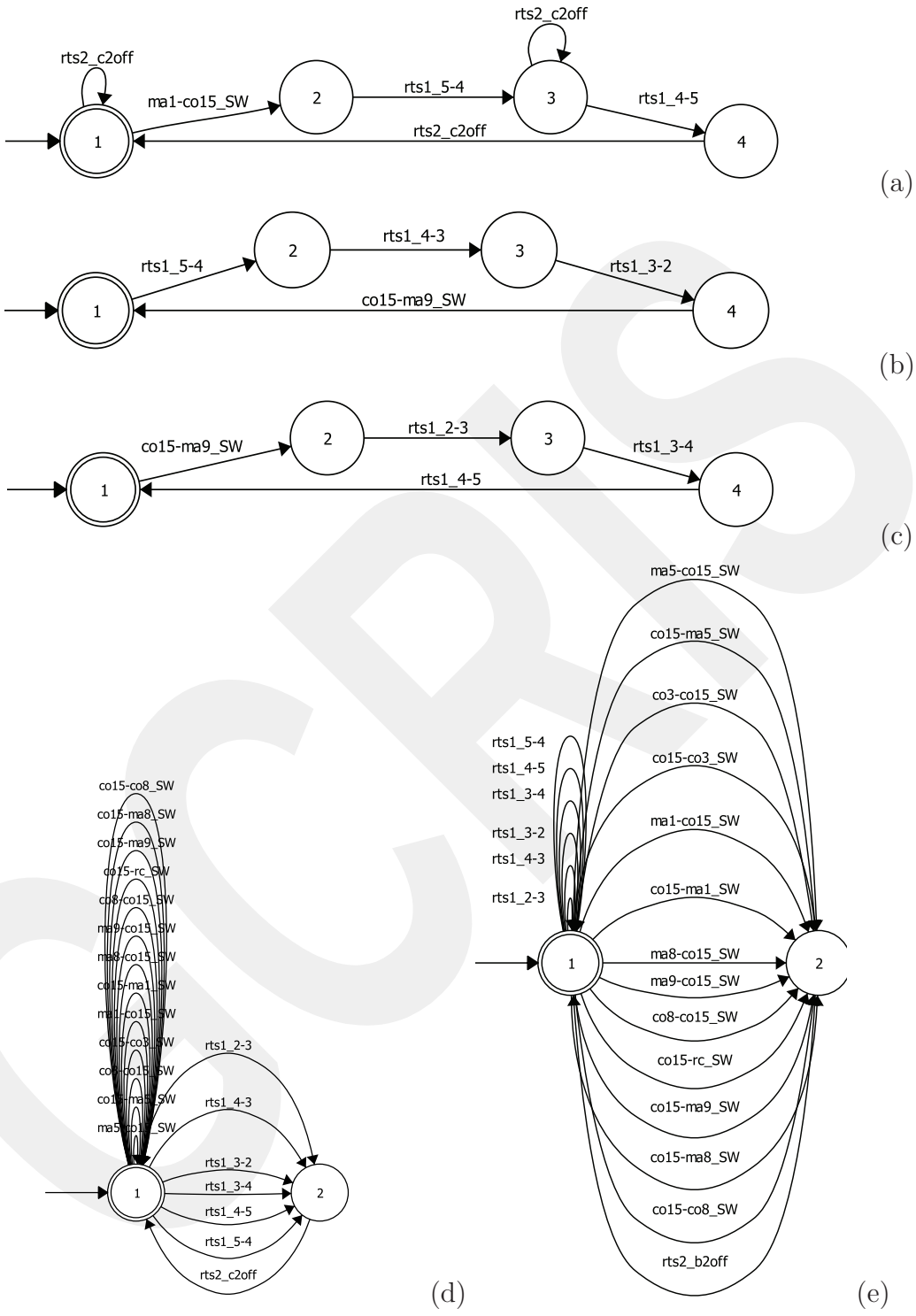

Figure 4.22: Module 4 configuration 2 specifications : (a), (b), (c), (d), (e).

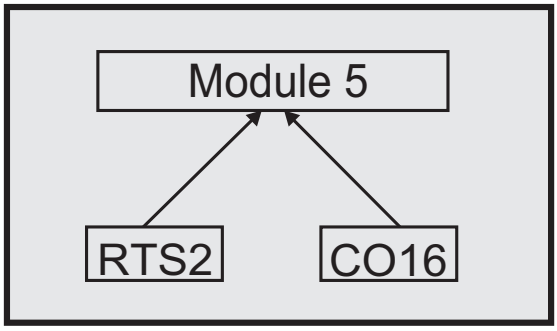

Figure 4.23: RMS module 5

In the second configuration only one path to move products from MA11 to RC is required. The specification for each configuration is computed by the synchronous composition of multiple small specifications that are listed as follows.

Configuration 1

- We consider the specification automaton $C_5^{1,1}$ in Figure 4.24 (a). *State 1 and state 2 describe path 1. Here, RTS2 moves from position 2 to position 3 after moving from position 1 to position 2. The second path is covered by states 1, 3, 4. Here, RTS2 moves to position 2 and transports products to RC if products move from MA11 to CO16.*

- *RTS2 should move from position 2 to position 3 first. Then it moves to position 4.* The corresponding specification automaton $C_5^{1,2}$ is shown in Figure 4.24 (b).

- *If RTS2 arrives at position 4, then products move from CO3 to CO16. After that, RTS2 moves to position 3.* The corresponding specification automaton $C_5^{1,3}$ is shown in Figure 4.24 (c).

- *Products move from CO16 to MA9, just after RTS2 arrives at position 2. If products move to MA9, then RTS2 moves from position 2 to 1.* The corresponding specification automaton $C_5^{1,4}$ is shown in Figure 4.24 (d).

- *Products first move to CO16 from MA11, then RTS2 moves to position 2.* The corresponding specification automaton $C_5^{1,5}$ is shown in Figure 4.24 (e).

- *After products leave RTS2 to RC, then RTS2 move´s back to its initial position.* The corresponding specification automaton $C_5^{1,6}$ is shown in Figure 4.24 (f).

- *If the rail transport system RTS2 is moving, then the conveyor belt CO16 should not move.* The corresponding specification automaton $C_5^{1,7}$ is shown in Figure 4.24 (g).

- *If the conveyor belt CO16 is moving products, then the rail transport system RTS2 should not move.* The corresponding specification automaton $C_5^{1,8}$ is shown in Figure 4.24 (h).

Configuration 2

- *If products moves from MA11 to CO16, then RTS2 moves to position 2 and stops. Afterwards, RTS2 moves back to position 1 and stops.* The corresponding specification automaton $C_5^{2,1}$ is shown in Figure 4.25 (a).

- *Products move to RC from CO16 just if RTS2 arrived at position 2. Furthermore, RTS2 moves from position 2 to position 1 just if products left CO16 to RC.* The corresponding specification automaton $C_5^{2,2}$ is shown in Figure 4.25 (b).

- *If the rail transport system RTS2 is moving, then the conveyor belt CO16 should not move.* The corresponding specification automaton $C_5^{2,3}$ is shown in Figure 4.25 (c).

- *If the conveyor belt CO16 is moving products, then the rail transport system RTS2 should not move.* The corresponding specification automaton $C_5^{2,4}$ is shown in Figure 4.25 (d).

Finally, the computational results for the configuration supervisors and reconfiguration supervisors are shown in Table 4.10.

Table 4.10: Module 5 reconfiguration Supervisors

SUPERVISORS	STATES	EVENTS	TRANSITIONS
S_5^1	22	20	39
S_5^2	8	20	12
$\hat{S}_5^1$	10	17	11
$\hat{S}_5^2$	4	17	4
R_5^1	45	26	111
R_5^2	17	26	55

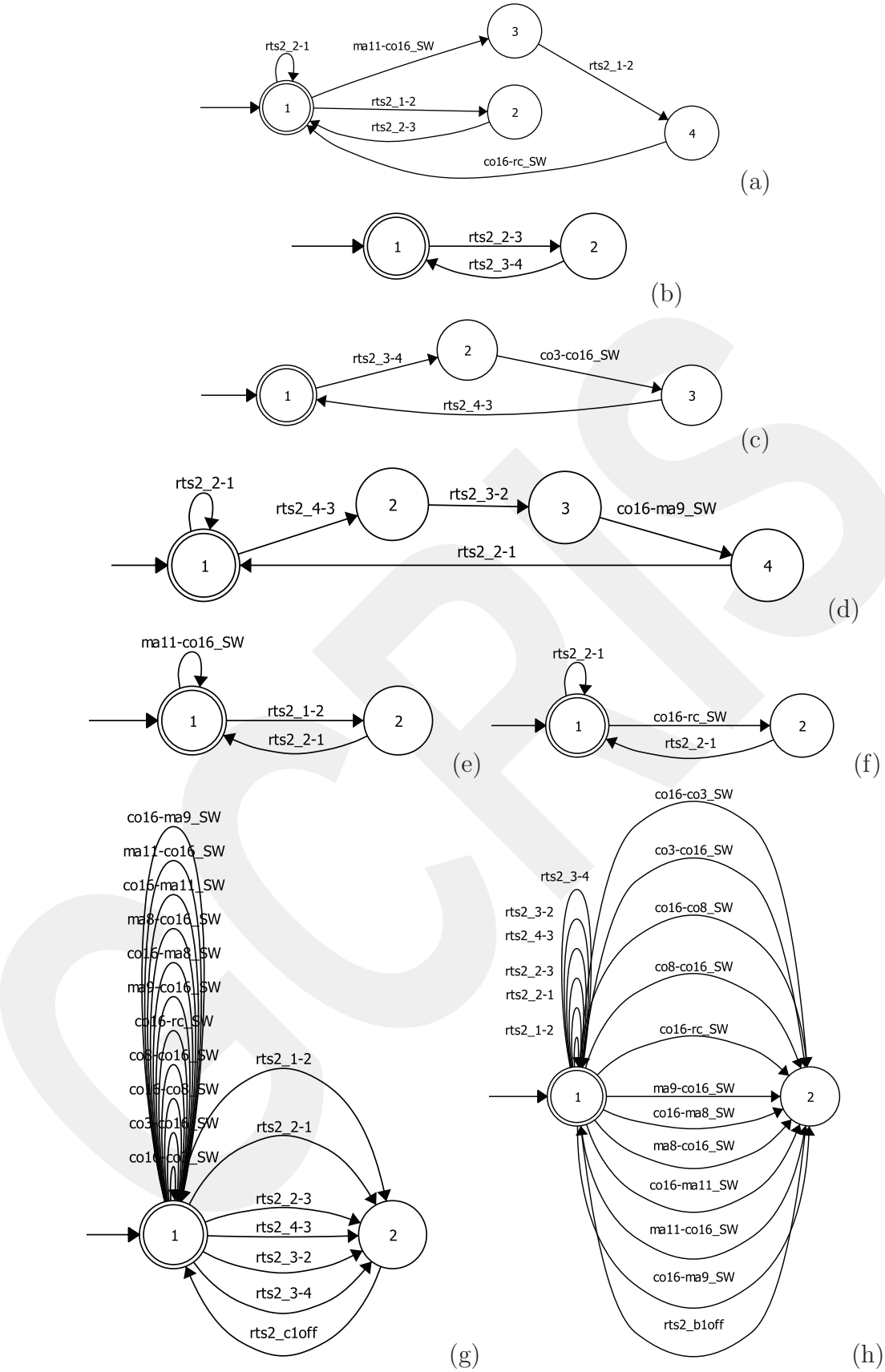

Figure 4.24: Module 5 configuration 1 specifications : (a), (b), (c), (d), (e), (f), (g), (h).

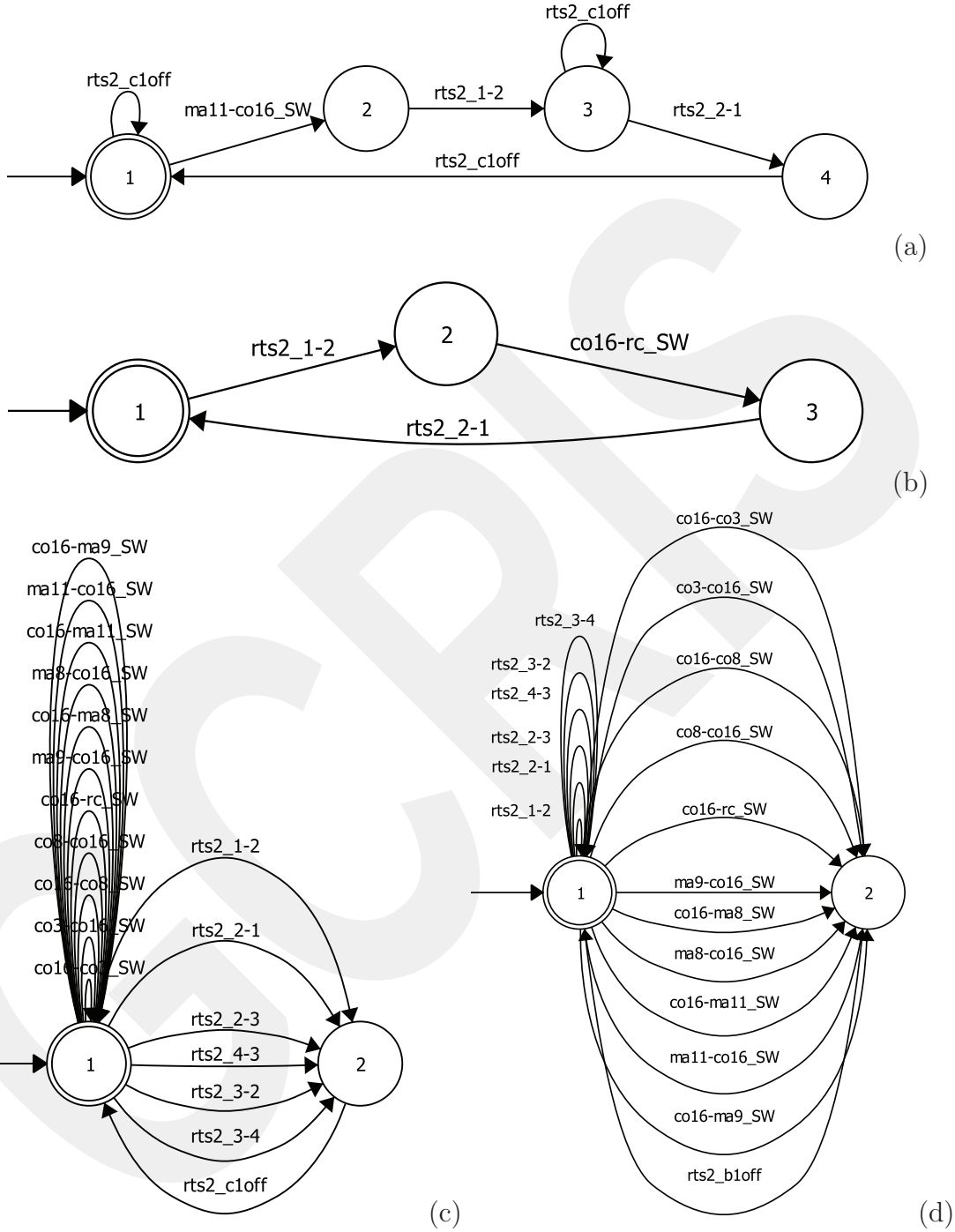

Figure 4.25: Module 5 configuration 2 specifications : (a), (b), (c), (d).

4.8 RMS MODULE 6

The single production machine MA9 is the only component of this module whose structure is shown in Figure 4.26. A product can enter to MA9 for processing, either from the rotary table RT6 on the upper side, or from RTS1 or RTS2 at the lower side of the machine. The abstracted model of MA9 is shown in Figure 4.27 and is denoted as

$$G_6 = G^{\text{high}}_{\text{MA9}} \tag{4.8}$$

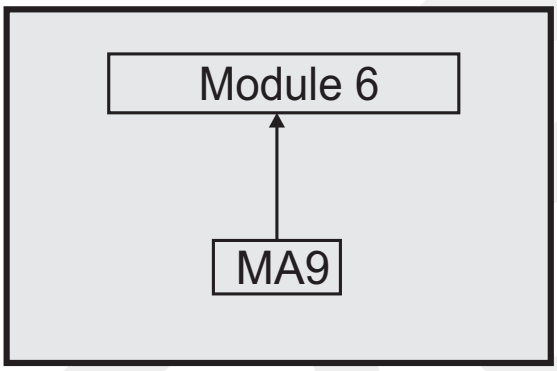

Figure 4.26: RMS module 6

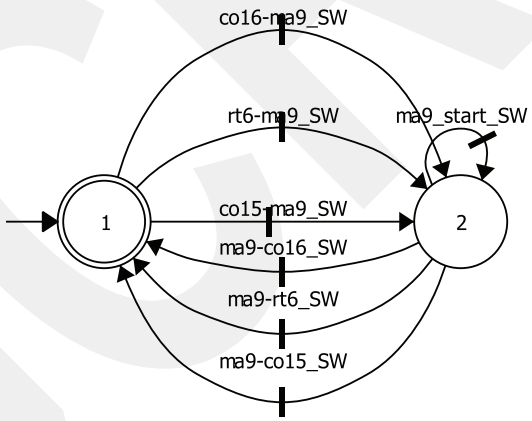

Figure 4.27: High level model of MA9

The operation of MA9 in the two configurations of the RMS are described as follows.

Configuration 1

- *If products arrive to MA9 from CO16, then the machine starts processing. After processing, products leave MA9 to RT6.* The corresponding specification automaton C_6^1 is shown in Figure 4.28.

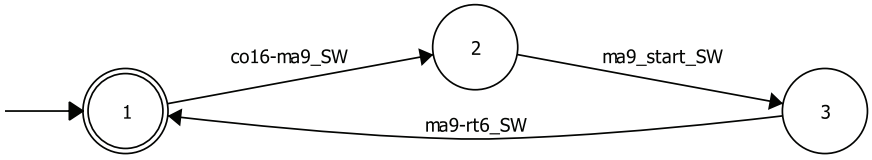

Figure 4.28: Module 6 configuration 1 specifications

Configuration 2

- *If products arrive at MA9, they are directly forwarded to RT6.* The corresponding specification automaton C_6^2 is shown in Figure 4.29.

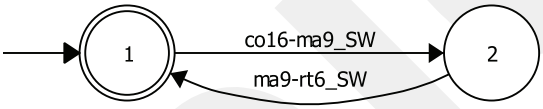

Figure 4.29: Module 6 configuration 2 specifications

In this case, the configuration supervisors are identical to the respective specification. The statistics are summarized in Table 4.11.

Table 4.11: Module 6 reconfiguration supervisors

SUPERVISORS	STATES	EVENTS	TRANSITIONS
S_6^1	3	7	3
S_6^2	2	7	2
$\hat{S}_6^1$	2	6	2
$\hat{S}_6^2$	2	6	2
R_6^1	7	13	23
R_6^2	5	13	20

4.9 RMS MODULE 7

This module consists of the rotary table RT6, the RMT MA10, the rotary table RT7, and the single production machine MA11. The structure of the module is shown in Figure 4.30. The abstracted plant models for these components are shown in Figure 4.31, Figure 4.32, Figure 4.33, and Figure 4.34. The overall plant

automaton

$$G_7 = G^{\text{high}}_{\text{RT6}} || G^{\text{high}}_{\text{MA10}} || G^{\text{high}}_{\text{RT7}} || G^{\text{high}}_{\text{MA11}} \tag{4.9}$$

has 64 states, 24 events, and 464 transitions. The operation of module 7 in the

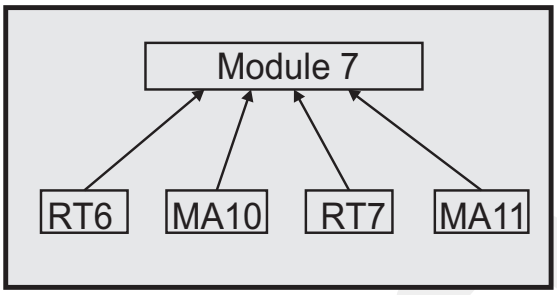

Figure 4.30: RMS module 7

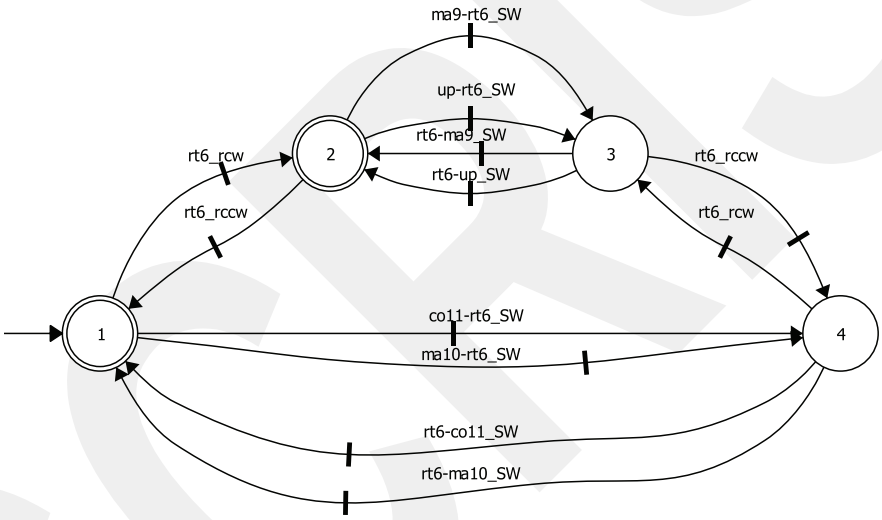

Figure 4.31: Abstracted RT6

two configurations of the RMS is described as follows.

Configuration 1

- *Products move from MA9 to RT6 after RT6 rotates in clockwise direction. After rotating back in counter-clockwise direction, products move to MA10.* The corresponding specification automaton $C_7^{1,1}$ is shown in Figure 4.35 (a).
- *If products arrived at MA10, then the machine starts processing. Afterwards, products move to RT7. If products arrived at RT7, then the rotary table rotates to deliver products to MA11.* The corresponding specification automaton $C_7^{1,2}$ is shown in Figure 4.35 (b).

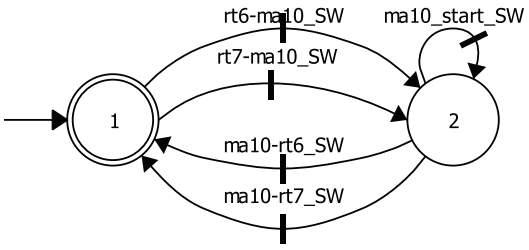

Figure 4.32: Abstracted MA10

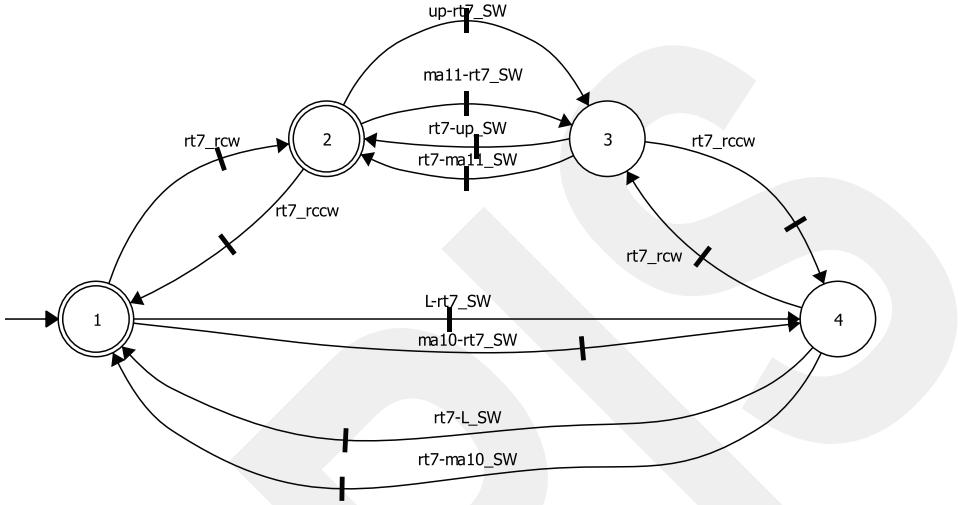

Figure 4.33: Abstracted RT7

- *If products arrived at MA11 from RT7, then they leave MA11 to RTS2.* The corresponding specification automaton $C_7^{1,3}$ is shown in Figure 4.35 (c).
- *If products were moved from RT7 to MA11 and RT7 is empty, then RT7 rotates back counter-clockwise.* The corresponding specification automaton $C_7^{1,4}$ is shown in Figure 4.35 (d).

Configuration 2

These specifications are analogous to the specifications for configuration 1. The only difference is that processing is performed by MA11 instead of MA10. The corresponding specifications automata $C_7^{2,1}$, $C_7^{2,2}$, $C_7^{2,3}$, and $C_7^{2,4}$ are shown in Figure 4.36 (a), (b), (c), (d).

The results of the supervisors for configuration 1 and 2 are summarized in Table 4.12.

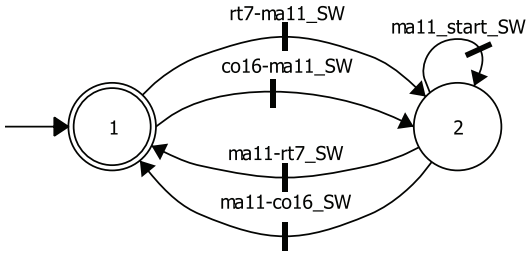

Figure 4.34: Abstracted MA11

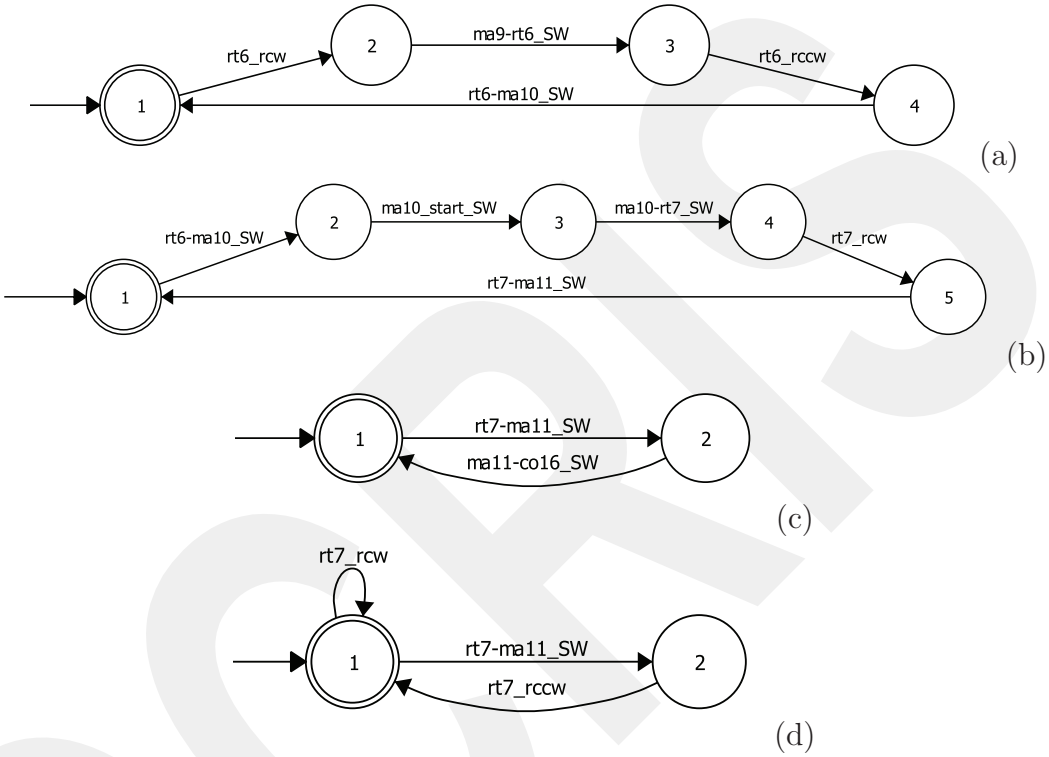

Figure 4.35: Module 7 configuration 1 specifications : (a), (b), (c), (d).

4.10 RMS MODULE 8

This module only consists of the exit slide RC.

$$G_8 = G_{\mathrm{RC}}^{\mathrm{high}} \tag{4.10}$$

The structure of the module is shown in Figure 4.37 and statistics are summarized in Table 4.13.

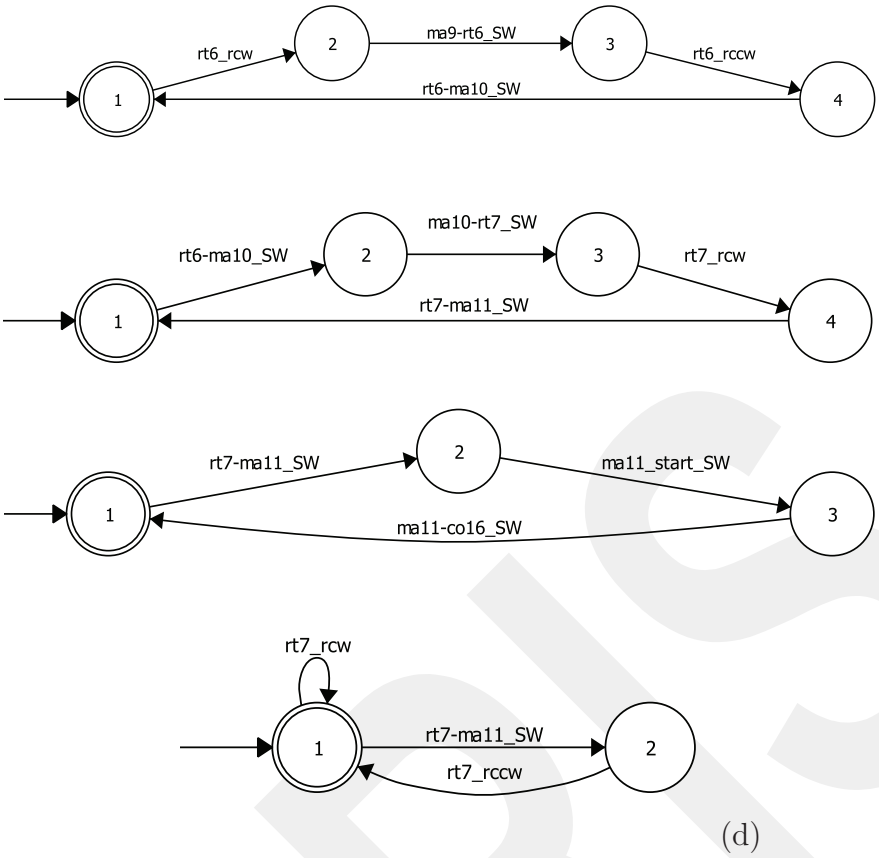

Figure 4.36: Module 7 configuration 2 specifications : (a), (b), (c), (d).

Table 4.12: Module 7 reconfiguration supervisors

SUPERVISORS	**STATES**	**EVENTS**	**TRANSITIONS**
S_7^1	28	24	51
S_7^2	36	24	70
$\hat{S}_7^1$	5	5	7
$\hat{S}_7^2$	5	5	7
R_7^1	57	30	164
R_7^2	73	30	210

Table 4.13: Module 8 configuration supervisors

SUPERVISORS	**STATES**	**EVENTS**	**TRANSITIONS**
S_8^1	3	4	4
S_8^2	3	4	4
$\hat{S}_8^1$	1	2	2
$\hat{S}_8^2$	1	2	2

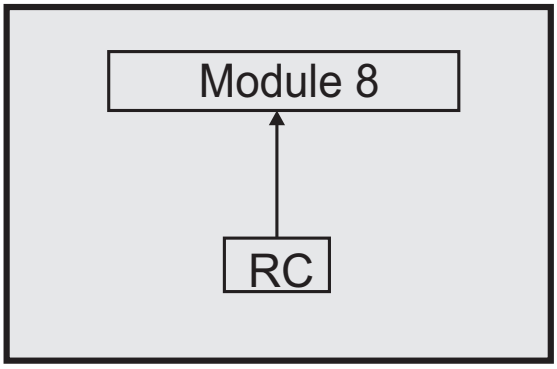

Figure 4.37: RMS module 8

Table 4.14: Module 8 reconfiguration supervisors

SUPERVISORS	STATES	EVENTS	TRANSITIONS
R_8^1	7	10	22
R_8^2	7	10	22

4.11 RMS HIGH-LEVEL MODULE 12

In this section, the RMS module 12 is introduced to describe the high-level control for module 1 and module 2 together as shown in Figure 4.38. The low-level configuration supervisors were computed before in chapter III, and are denoted as S_1^1 and S_1^2 respectively for the two configurations of module 1. Similarly, S_2^1 and S_2^2 were computed for configuration 1 and 2 of module 2. Furthermore, the abstraction-based approach was applied to these supervisors with the result $\hat{S}_1^1$ and $\hat{S}_1^2$ for module 1, and $\hat{S}_2^1$, $\hat{S}_2^2$ for module 2. For the plant of this module,

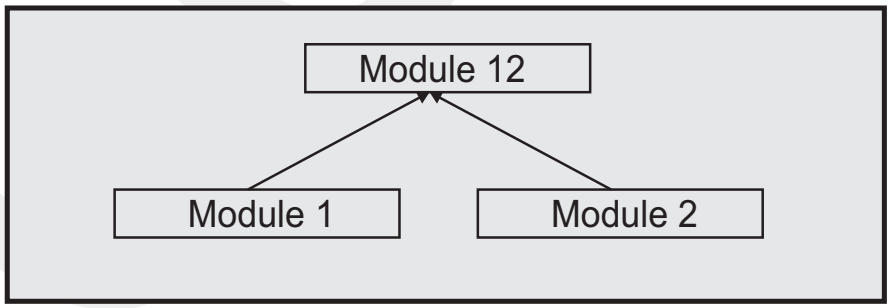

Figure 4.38: RMS high-level module 12

that is realized by the synchronous composition of the high-level abstractions, we introduce G_{12}^1 for the first configuration G_{12}^2 for the second configuration. These automata computed as

$$G_{12}^1 = \hat{S}_1^1 || \hat{S}_2^1 \tag{4.11}$$

with 6 states, 7 events, and 9 transitions and

$$G_{12}^2 = \hat{S}_1^2 || \hat{S}_2^2 \tag{4.12}$$

with 3 states, 7 events, and 4 transitions.

In order to compute a supervisor for the two configurations of module 12, we note that the desired behavior was already achieved by the supervisor computation for module 1 and module 2. Here, it is only required to make module 12 nonblocking, which is achieved by using the nonblocking part of G_{12} as a specification for module 12. The statistics of the resulting supervisor computation are summarized in Table 4.15. In addition, we need the coordinating automata P_{12}^1 and P_{12}^2 respectively in Figure 4.39 and Figure 4.40 to complete the reconfiguration.

Table 4.15: Module 12 reconfiguration supervisors

SUPERVISORS	STATES	EVENTS	TRANSITIONS
S_{12}^1	6	7	9
S_{12}^2	3	7	4
$\hat{S}_{12}^1$	4	5	6
$\hat{S}_{12}^2$	3	5	4
R_{12}^1	13	10	39
R_{12}^2	7	10	26

4.12 RMS HIGH-LEVEL MODULE 3456

This module is the high-level of multiple modules, module 3, module 4, module 5, and module 6. The structure of this module is shown in Figure 4.41, t The abstractions $\hat{S}_3^1$ and $\hat{S}_3^2$ for module 3, $\hat{S}_4^1$, $\hat{S}_4^2$ for module 4, $\hat{S}_5^1$ and $\hat{S}_5^2$ for module 5 and $\hat{S}_6^1$, $\hat{S}_6^2$ for module 6 are used to compute the plant automata G_{3456}^1 (configuration 1) and G_{3456}^2 (configuration 2):

$$G_{3456}^1 = \hat{S}_3^1 || \hat{S}_4^1 || \hat{S}_5^1 || \hat{S}_6^1 \tag{4.13}$$

$$G_{3456}^2 = \hat{S}_3^2 || \hat{S}_4^2 || \hat{S}_5^2 || \hat{S}_6^2 \tag{4.14}$$

As a result, G_{3456}^1 has 40 states, 40 events, and 80 transitions, and G_{3456}^2 has 64 states, 40 events, and 156 transitions. We use the following specifications for module 3456.

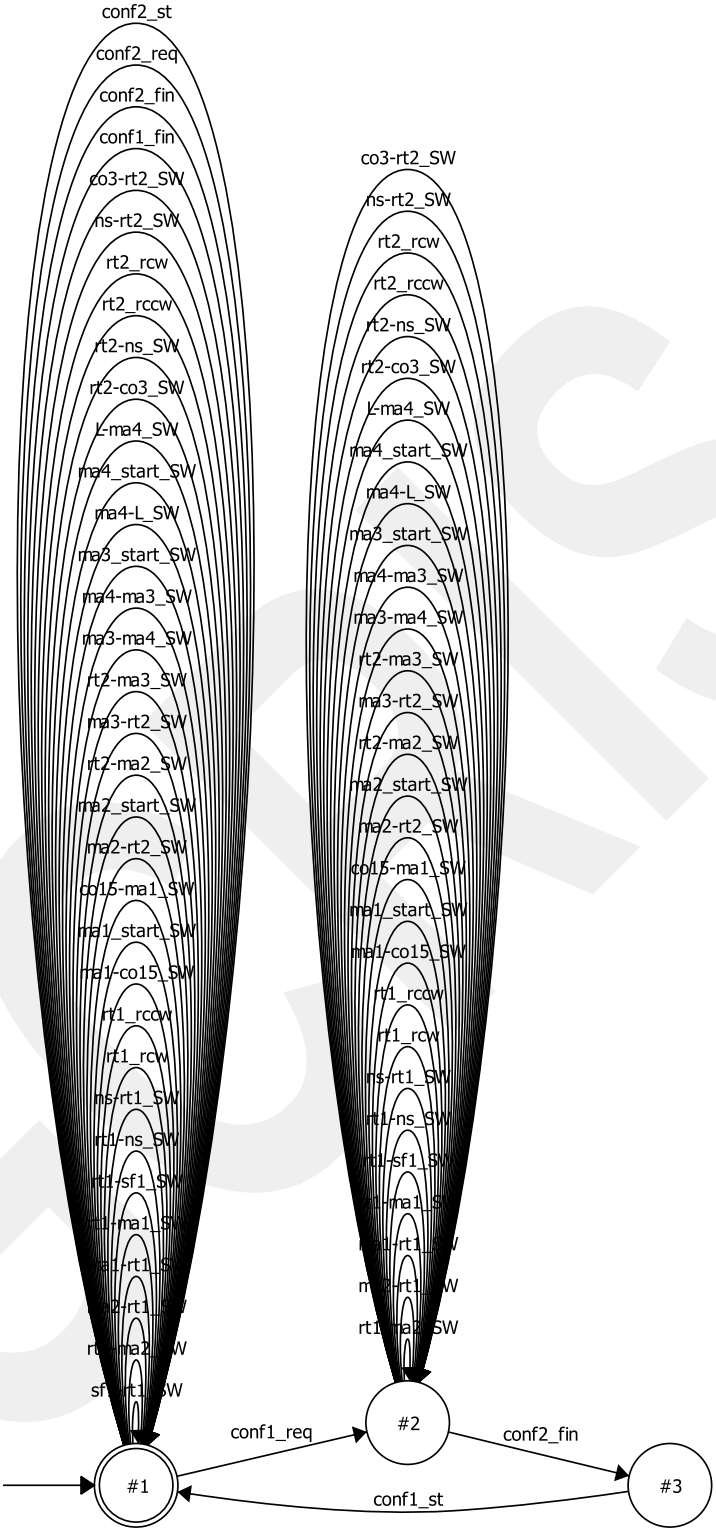

Figure 4.39: P_{12}^{1} coordination automaton

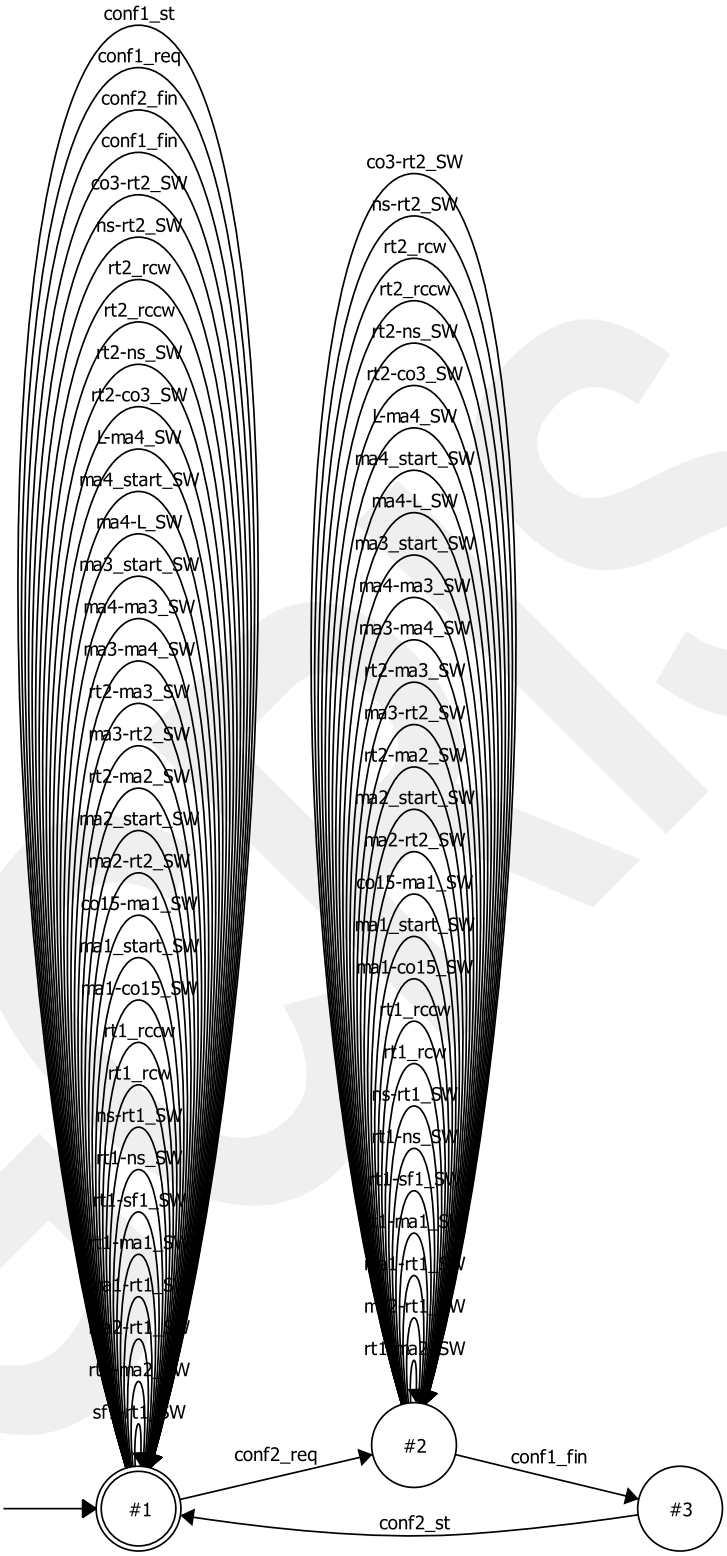

Figure 4.40: P_{12}^2 coordination automaton

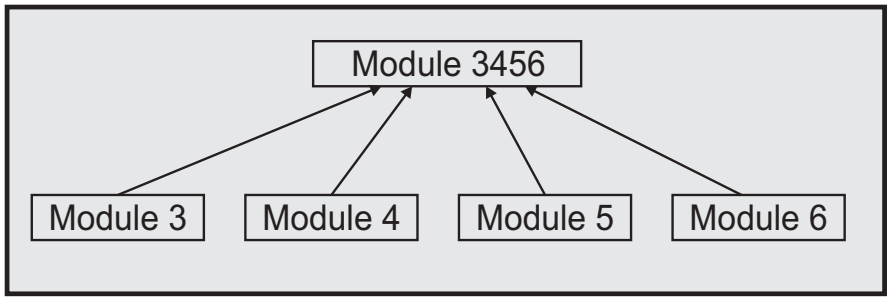

Figure 4.41: RMS high-level module 3456

Configuration 1

- *If products were transported from RT2 to CO3, then RTS2 moves from position 1 to 2. Then, CO3 moves products to CO16. In addition, if MA11 moves products to RTS2 then RTS2 moves to position 2, and products are transported to RC. Then, RTS2 moves back to position 1.* The corresponding specification automaton $C_{3456}^{1,1}$ is shown in Figure 4.42 (a).
- *If CO3 moves products to CO16, then CO16 moves them to MA9.* The corresponding specification automaton $C_{3456}^{1,2}$ is shown in Figure 4.42 (b).
- *If CO16 moves products to MA9, then MA9 transports products to RT6.* The corresponding specification automaton $C_{3456}^{1,3}$ is shown in Figure 4.42 (c).
- *Just if products were moved from CO16 to MA9, then RTS2 moves from position 2 to position 1.* The corresponding specification automaton $C_{3456}^{1,4}$ is shown in Figure 4.42 (d).

Configuration 2

- *If Ma11 moves products to CO16, then CO16 moves products to RC.* The corresponding specification automaton $C_{3456}^{2,1}$ is shown in Figure 4.43 (a).
- *If CO15 moves products to MA9, then MA9 moves products to RT6.* The corresponding specification automaton $C_{3456}^{2,2}$ is shown in Figure 4.43 (b).
- *Just if products were moved from MA1 to CO15, then RTS1 moves from position 5 to position 4. If RTS1 arrives at position 4, then it moves back to position 5.* The corresponding specification automaton $C_{3456}^{2,3}$ is shown in Figure 4.43 (c).

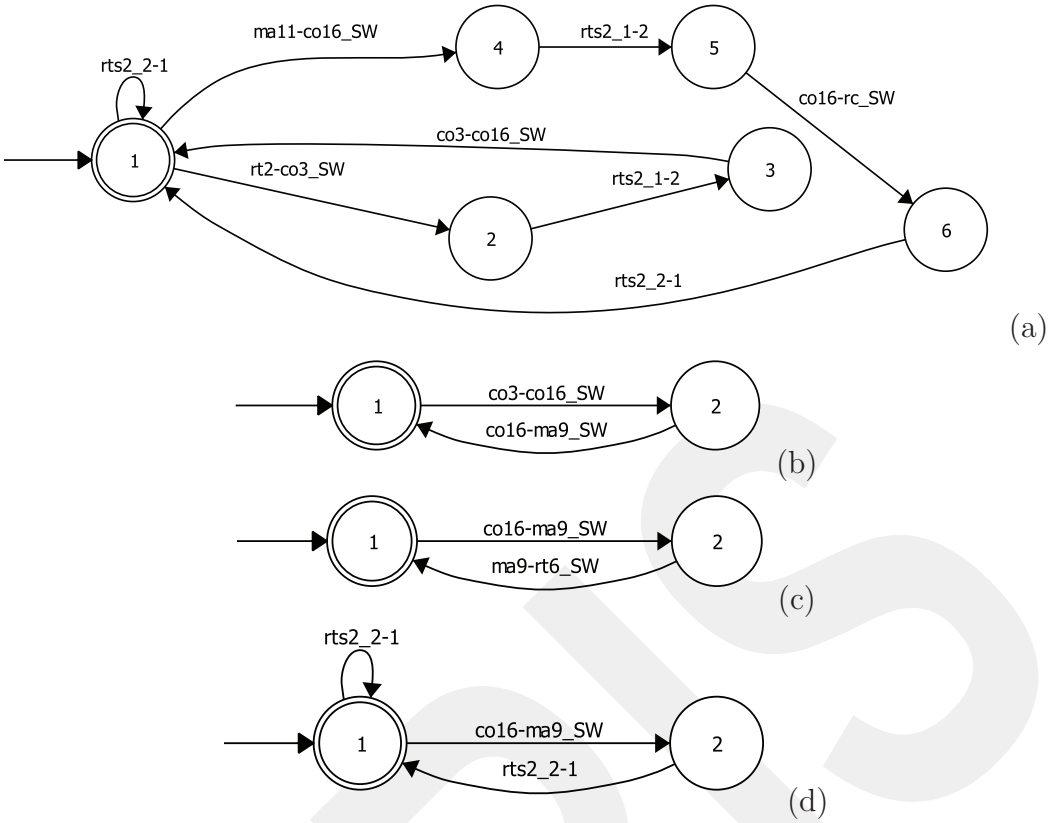

Figure 4.42: Module 3456 configuration 1 specifications : (a), (b), (c), (d).

- *If Ma1 moves products to CO15, then CO15 moves products to MA9.* The corresponding specification automaton $C_{3456}^{2,4}$ is shown in Figure 4.43 (d).
- *Just if products move from CO15 to MA9, then RTS1 moves from position 2 to position 3.* The corresponding specification automaton $C_{3456}^{2,5}$ is shown in Figure 4.43 (e).
- To prevent crash in position 2, *if RTS2 moves from position 1 to 2, then RTS1 can move from position 3 to 2 only if RTS2 was moved back from position 2 to 1, and vice versa.* The corresponding specification automaton $C_{3456}^{2,6}$ is shown in Figure 4.43 (f).
- To prevent crash in position 3, *if RTS2 moves from position 2 to 3, then RTS1 can move from position 4 to 3 only if RTS2 was moved back from position 3 to 2, and vice versa.* The corresponding specification automaton $C_{3456}^{2,7}$ is shown in Figure 4.43 (g).
- To prevent crash in position 4, *if RTS2 moves from position 3 to 4, then RTS1 can move from position 5 to 4 only if RTS2 was moved back from*

position 4 to 3, and vice versa. The corresponding specification automaton $C^{2,8}_{3456}$ is shown in Figure 4.43 (h).

Finally, we use the *SupCon* algorithm for the plants and the related specifications to compute the configuration supervisors for this module. The results are shown in Table 4.16. The coordinating automata P^1_{3456} and P^2_{3456} are shown in Figure 4.44 and Figure 4.45.

Table 4.16: Module 3456 reconfiguration supervisors

SUPERVISORS	STATES	EVENTS	TRANSITIONS
S^1_{3456}	24	40	37
S^2_{3456}	56	40	130
$\hat{S}^1_{3456}$	5	20	9
$\hat{S}^2_{3456}$	14	22	31
R^1_{3456}	49	46	145
R^2_{3456}	113	46	363

4.13 RMS HIGH-LEVEL MODULE 123456

We introduce module 123456 as the high-level for module 12 and module 3456. The structure of the module is shown in Figure 4.46. The model of the plant for the first configuration is denoted as G^1_{123456} and the model of the second configuration is G^2_{123456}. Hereby, G^1_{123456} is obtained for configuration 1 by the synchronous composition of the abstraction configuration supervisors $\hat{S}^1_{12}$ with 4 states, 5 events and 6 transitions and $\hat{S}^1_{3456}$ with 3 states, 5 events and 4 transitions. In addition, G^2_{123456} for configuration 2, is obtained by the synchronous composition of the abstraction configuration supervisors $\hat{S}^2_{12}$ with 5 states, 20 events and 9 transitions, and $\hat{S}^2_{3456}$ with 14 states, 22 events, and 31 transitions. The overall model of the plant for both configurations is obtained as

$$G^1_{123456} = \hat{S}^1_{12} || \hat{S}^1_{3456} \tag{4.15}$$

$$G^2_{123456} = \hat{S}^2_{12} || \hat{S}^2_{3456} \tag{4.16}$$

For the computation of the configuration supervisors for configuration 1 and configuration 2, we use the nonblocking part of the plant as the specification. The results of this computation are summarized in Table 4.17.

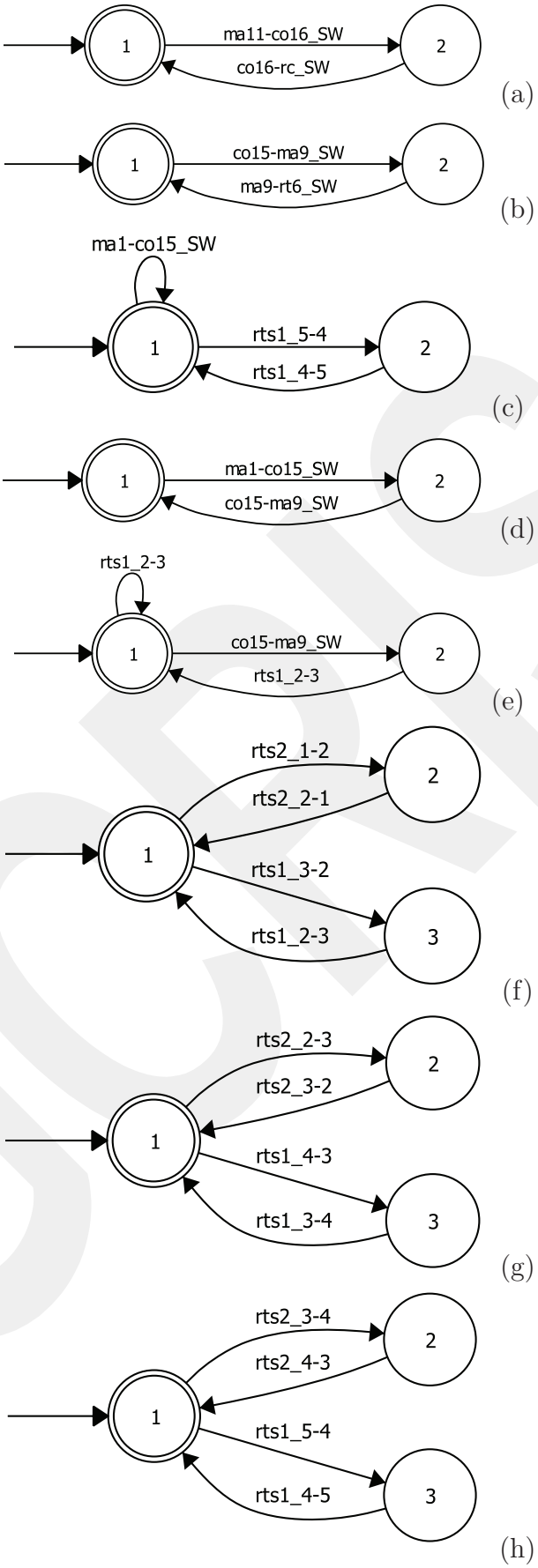

Figure 4.43: Module 3456 configuration 2 specifications: (a), (b), (c), (d), (e), (f), (g), (h).

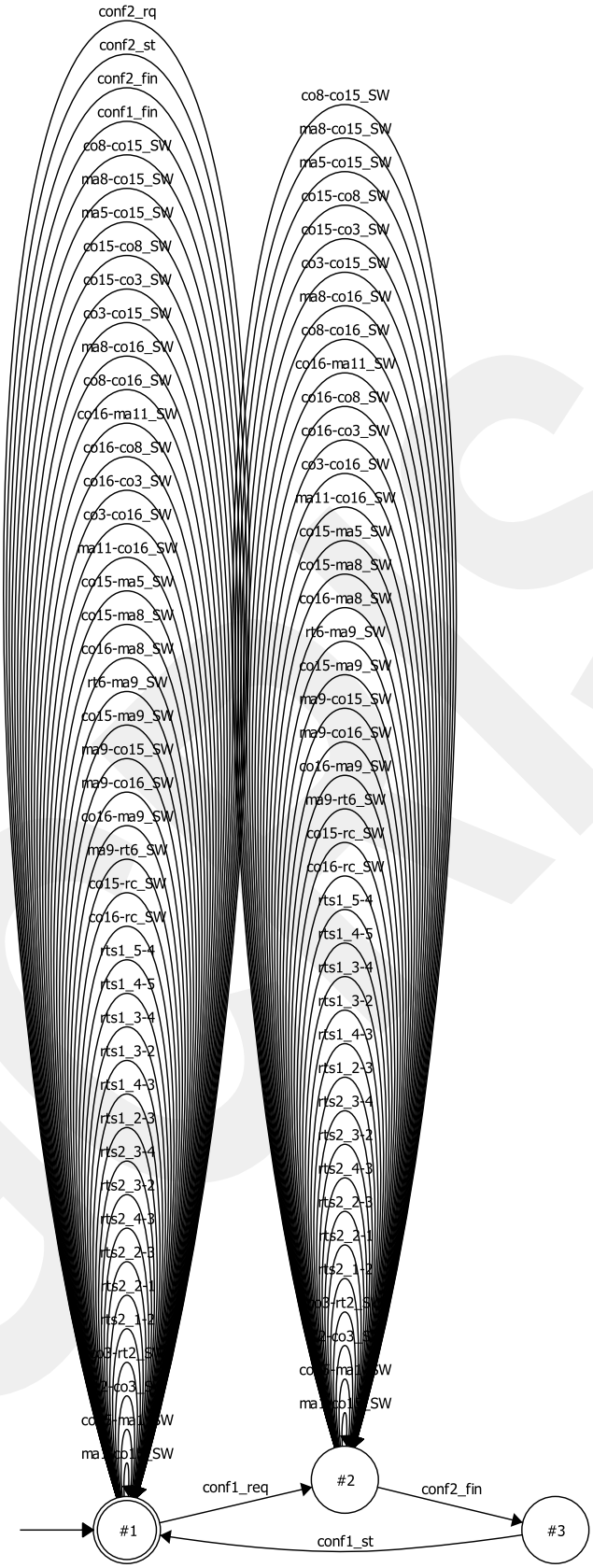

Figure 4.44: P^1_{3456} coordination automaton

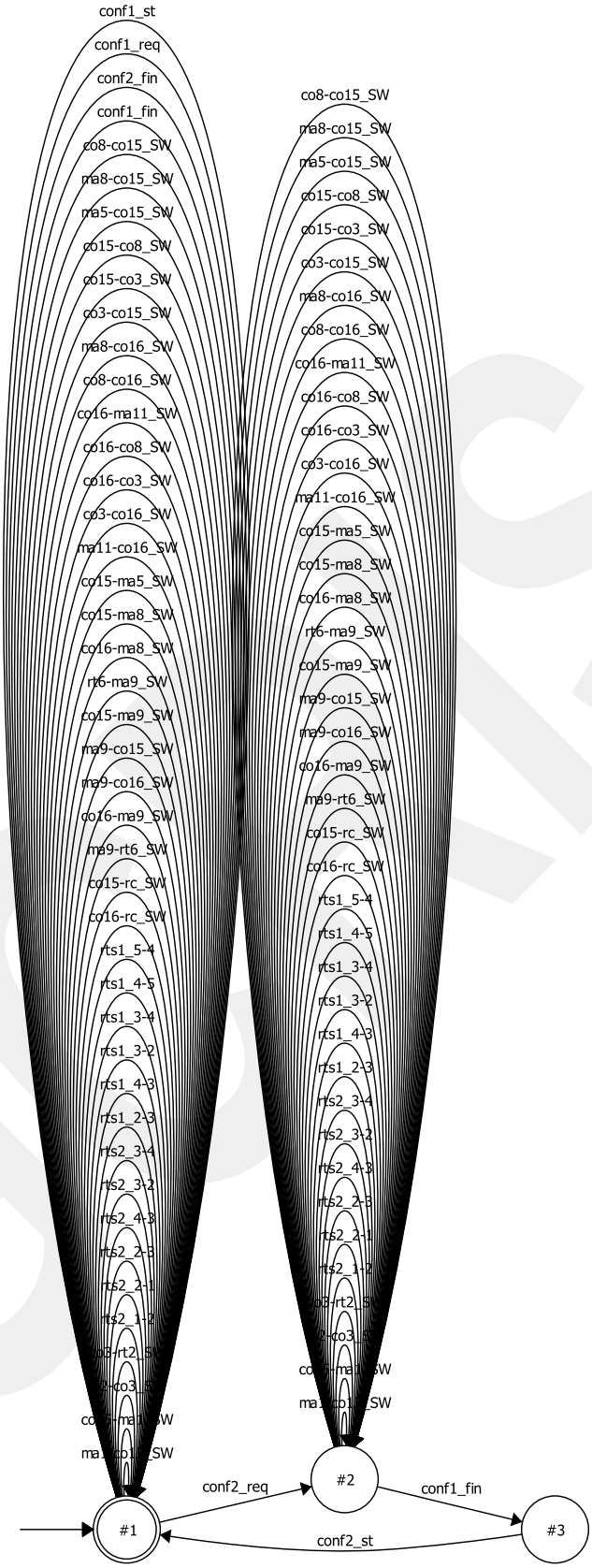

Figure 4.45: P^2_{3456} coordination automaton

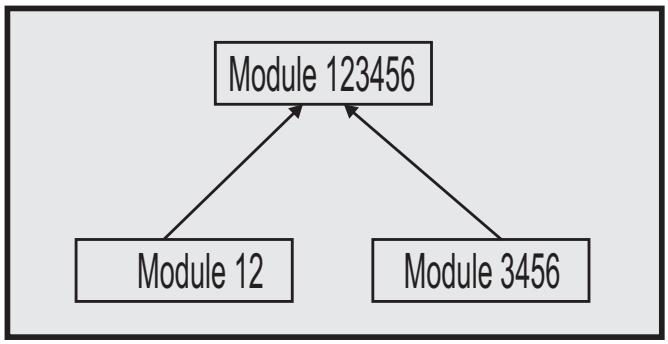

Figure 4.46: RMS high-level module 123456

Table 4.17: Module 123456 reconfiguration supervisors

SUPERVISORS	STATES	EVENTS	TRANSITIONS
S^1_{123456}	20	21	49
S^2_{123456}	42	23	115
R^1_{123456}	41	27	146
R^2_{123456}	85	29	302

4.14 RMS HIGH-LEVEL MODULE 12345678

The high-level module for the overall RMS is given by this module whose structure is shown in Figure 4.47. It is composed of module 123456, module 7 and module 8. We compute the overall plant for the first and second configuration using $\hat{S}^1_{123456}$, $\hat{S}^1_7$, and $\hat{S}^1_8$ as

$$G^1_{12345678} = \hat{S}^1_{123456} || \hat{S}^1_7 || \hat{S}^1_8 \tag{4.17}$$

$$G^2_{12345678} = \hat{S}^2_{123456} || \hat{S}^2_7 || \hat{S}^2_8 \tag{4.18}$$

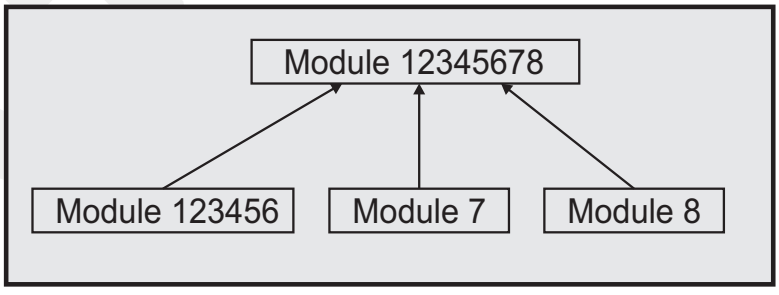

Figure 4.47: RMS high-level module 12345678

The overall plant automata is too big to shown here. It has 96 states, 9 events, and 226 transitions for configuration 1, and it has 150 states, 10 events, and 379

transitions for configuration 2. For this module, we use a specification in configuration configuration 1 such that RT6 should only rotate if products are moved from RT2 to CO3. Similarly, for configuration 2, we want that RT6 should only rotate if RTS1 is moved from position 3 to position 2, as is shown in Figure 4.48 and Figure 4.49.

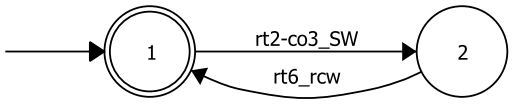

Figure 4.48: Module 12345678 configuration 1 specifications

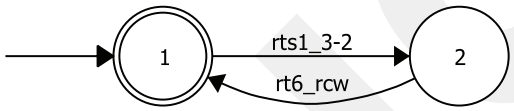

Figure 4.49: Module 12345678 configuration 2 specifications

The overall configuration supervisors are computed according to the above specifications. The results are summarized in Table 4.18. $P^1_{12345678}$ and $P^2_{12345678}$ are the coordinating automata for this module as shown in Figure 4.50 and Figure 4.51.

Table 4.18: Module 12345678 reconfiguration supervisors

SUPERVISORS	STATES	EVENTS	TRANSITIONS
$S^1_{12345678}$	68	9	154
$S^2_{12345678}$	108	10	257
$R^1_{12345678}$	137	15	341
$R^2_{12345678}$	217	16	559

4.15 SUMMARY OF ABSTRACTION-BASED RECONFIGURATION SUPERVISORS

In the previous sections, we computed reconfiguration supervisors for all modules of the RMS. The hierarchical structure of the overall supervisor design is shown in Figure 4.52. It shows that our design requires 4 levels with a total number of 12 supervisors on these levels. The overall reconfiguration supervisor for the RMS is then given by the synchronous composition of these supervisors. Note

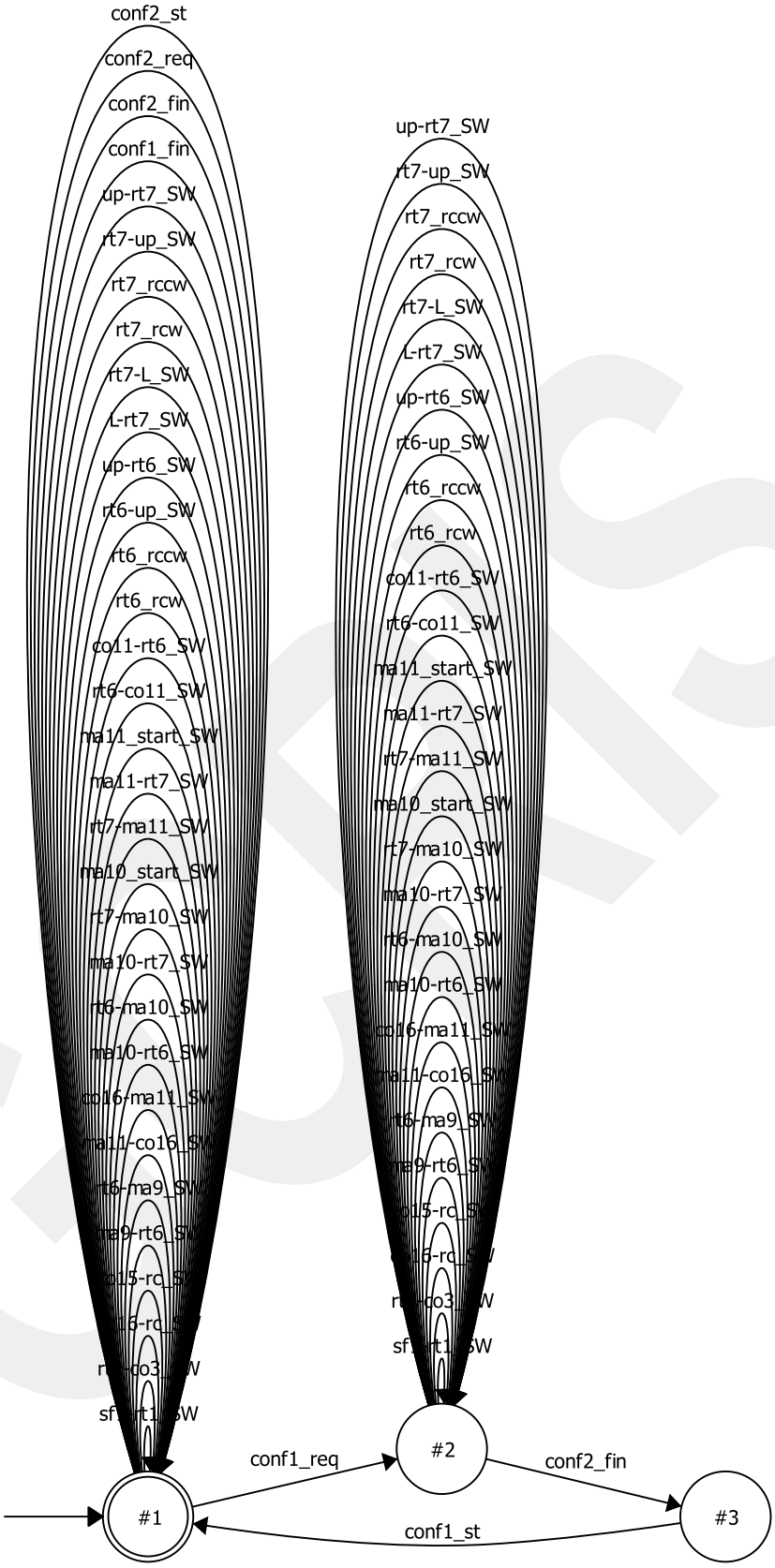

Figure 4.50: $P^1_{12345678}$ automata

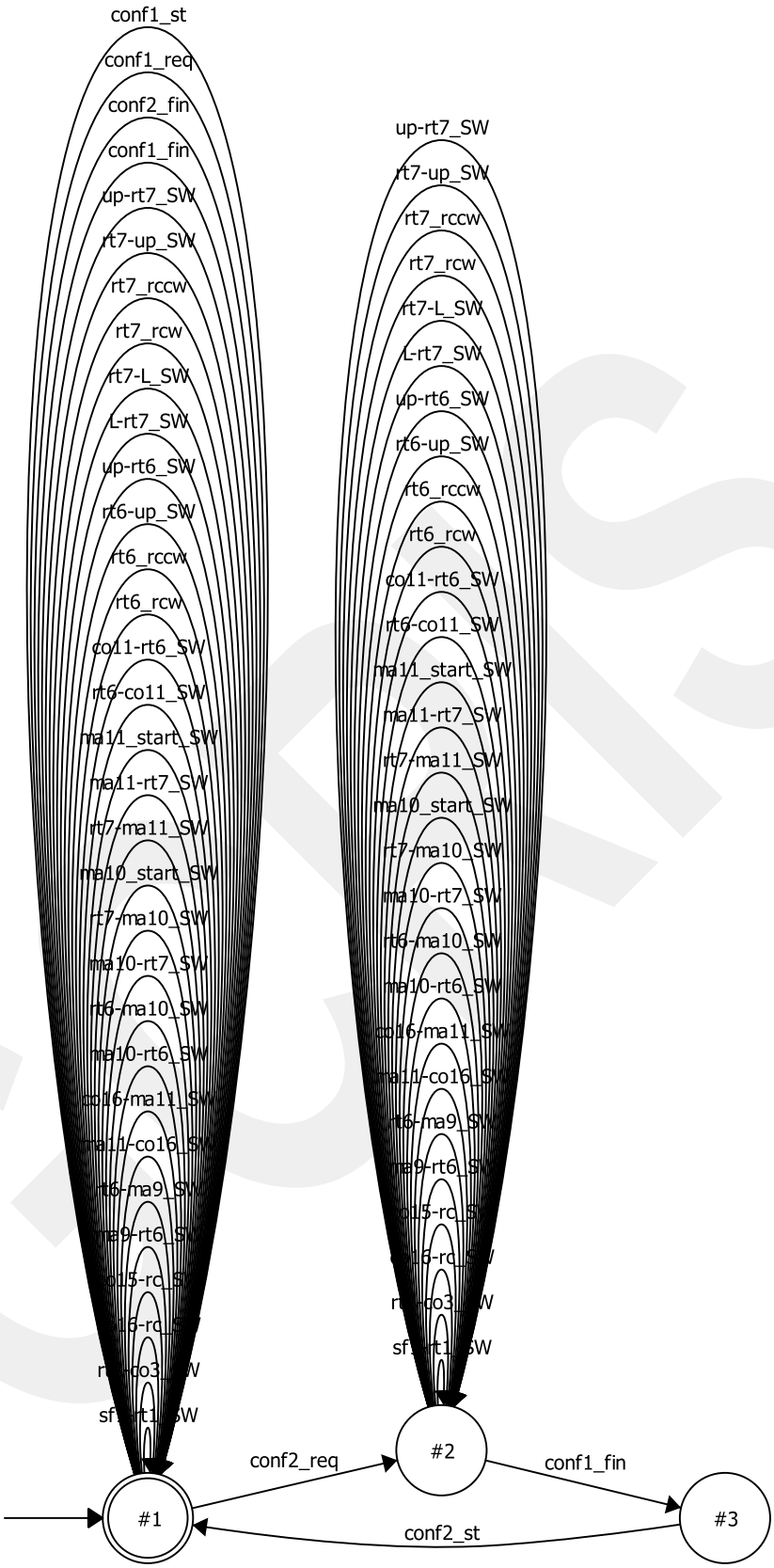

Figure 4.51: $P^2_{12345678}$ automata

however, that this synchronous composition does not need to be computed for a controller implementation. In a practical implementation, it is sufficient to run all supervisors in parallel. Hence, the overall state size of the RMS reconfiguration supervisor is given by the sum of the state sizes of the module supervisors which is in the order of 1000 states.

Here, the advantage of the abstraction-based design can be seen if the obtained result is compared to a monolithic design (computation for a single plant model of the RMS). In that case, a supervisor with an estimated state size of 10^{28} states would be needed which is computationally infeasible and cannot be realized in any physical controller device.

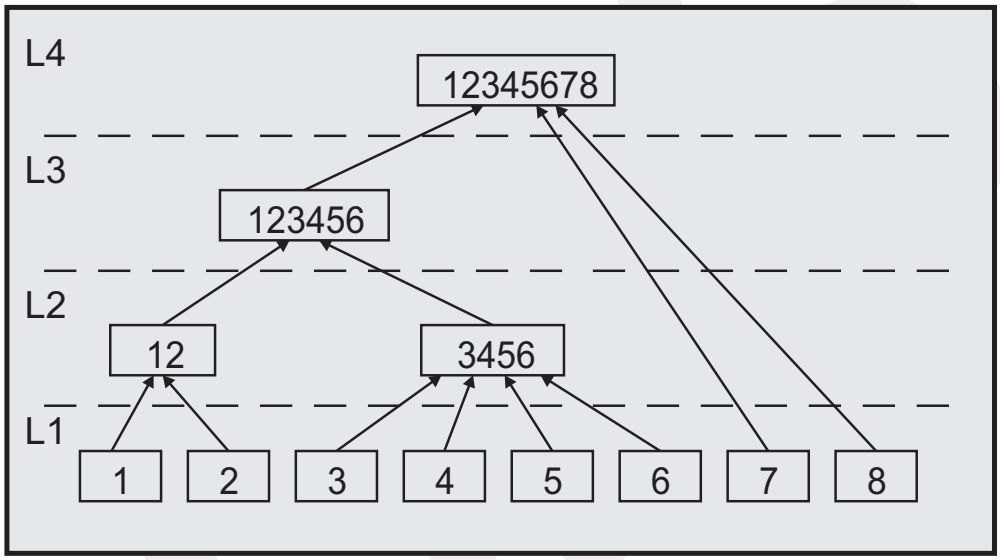

Figure 4.52: Hierarchical and decentralized RMS control

4.16 RMS SIMULATION

In addition to the controller computation, a simulation model of the RMS example is developed. This simulation is based on the software FlexFact for the simulation of the physical RMS and the software DESTool for the simulation of the controller automata. In addition, a realt-time simulation of the controller automata with the software simfaudes is performed. All software tools are distributed together with the libfaudes software library for DES [14].

A screenshot of the simulation environment is shown in Figure 4.53. On the upper side, the FlexFact graphical user interface is shown. It represents the RMS plant

behavior as described before. The lower side shows DESTool which simulates the DES controllers.

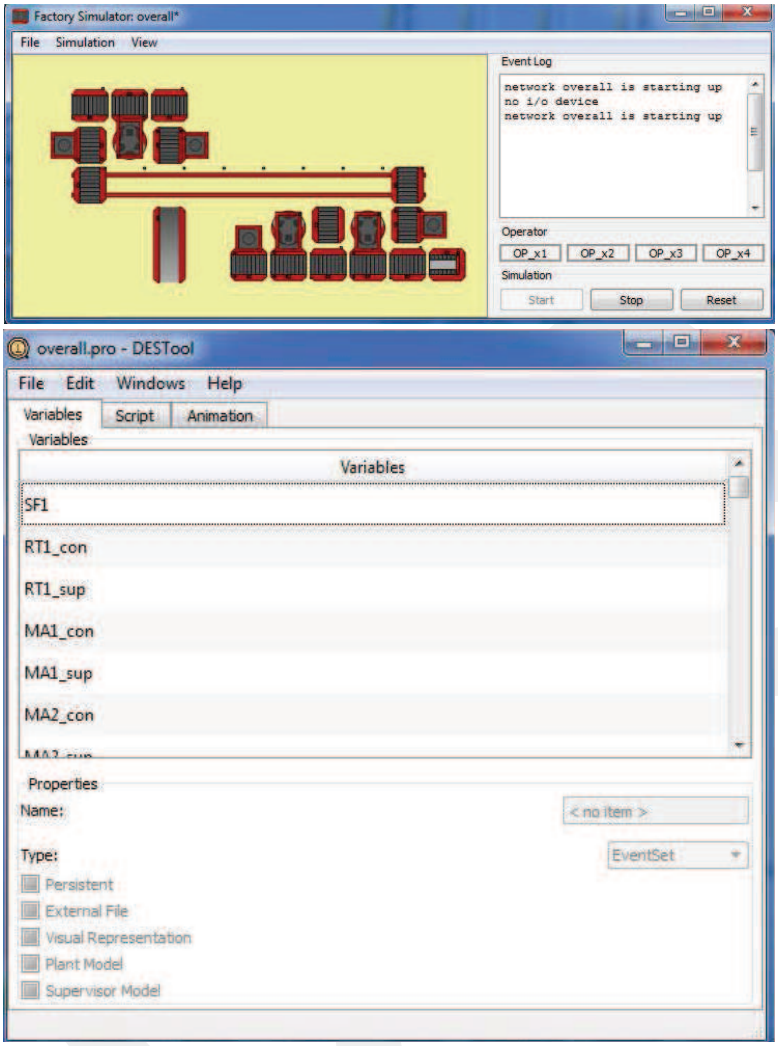

Figure 4.53: RMS simulation

The overall supervisory control loop for the RMS example in this chapter was implemented in FlexFact and DESTool and the correctness of the supervisor computation was confirmed by extensive simulation studies.

CONCLUSION

In this thesis, we consider the supervisory control of large-scale reconfigurable manufacturing systems (RMS). RMS are used for the fast response to variable changes in product types and product volumes that are prominent in today's manufacturing industry. As an important system component, RMS include reconfigurable machine tools (RMT) that are able to perform multiple operations depending on their active configuration.

In principle, the control design for RMS should make it possible to change between different system configurations at any time. If such change is requested, it is desired to first complete the currently active configuration before starting up the newly requested configuration. In each such change, both the manufacturing processes of all RMS components must be finished and in each start-up, the configuration of each RMT has to be adjusted according to the desired operation in the new configuration. Such reconfiguration design should be possible both for medium size as well as large-scale RMS.

All the described issues are addressed in this thesis in the framework of supervisory control for discrete event systems (DES). Hereby, the developed design method is based on the idea of abstraction-based supervisory control. First, a modeling framework for RMTs is introduced and applied to an example RMT. The advantage of the modeling framework is that arbitrary configuration changes of the RMT are possible, whereas the resulting RMT model is represented by an automaton with a small number of states. Second, we describe a new algorithms for the construction of reconfiguration supervisors for RMSs. This algorithm suggests the computation of a modular reconfiguration supervisor for each configuration. Each such supervisor realizes the desired operation of the respective configuration and allows completing each configuration before starting a new configuration based on the idea of state attraction. In addition, we extend the framework to the application of abstraction-based control in order to enable the use of small system models. An important advantage of the proposed method is its application in supervisor hierarchies with an arbitrary number of levels. The applicability of the

developed method is demonstrated by a large-scale laboratory RMS with 18 system components. It is shown that the resulting supervisors have a maximum state size of 217 states, whereas a classical monolithic design would require a state size in the order of 10^{28} states. The correctness of the design method is also verified by a simulation study of the laboratory RMS.

FUTURE WORK

In future work, it is desired to apply the designed reconfiguration supervisors to the real laboratory system by using the Libfaudes simulator plug-in and the IOdevice plug-in.

In addition, the computation of the modified state attractors is currently under investigation.

A further observation is that, in the current design, the start-up of a new configuration has to wait until the previous configuration is completed. It is envisaged to perform this start-up as early as possible without waiting for the completion of the previous configuration. A first result in this direction is given by [22].

REFERENCES

[1] Y. Brave and M. Heymann, "Stabilization of discrete-event processes," *Int. J. Control*, vol. 51, pp. 1101–1117, 1990.

[2] Y. Brave and M. Heymann, "On optimal attraction of discrete-event processes," *Information Sciences*, vol. 67, pp. 245–276, 1993.

[3] A. Dashchenko (Ed.), *Reconfigurable manufacturing systems and transformable factories.* Springer, 2006.

[4] H. A. ElMaraghy, *Changeable and Reconfigurable Manufacturing Systems.* Springer Series in Advanced Manufacturing, 2009.

[5] E. W. Endsley, E. E. Almeida, and D. M. Tilbury, "Modular finite state machines: Development and application to reconfigurable manufacturing cell controller generation," *Control Engineering Practice*, vol. 14, no. 10, pp. 1127–1142, 2006.

[6] G. Faraut, L. Piétrac, and E. Niel, "Formal approach to multimodal control design: Application to mode switching," *Industrial Informatics, IEEE Transactions on*, vol. 5, no. 4, pp. 443–453, 2009.

[7] H. E. Garcia and A. Ray, "State-space supervisory control of reconfigurable discrete event systems," *Int. J. Control*, vol. 63, no. 4, pp. 767–797, 1996.

[8] R. Harrison, A. Colombo, A. West, and S. Lee, "Reconfigurable modular automation systems for automotive power-train manufacture," *International Journal of Flexible Manufacturing Systems*, vol. 18, no. 3, pp. 175–190, Sep. 2006.

[9] Harith M. Khalid, M. S. Kırık, and K. W. Schmidt, "Abstraction-based supervisory control for reconfigurable manufacturing systems," Workshop on *Dependable Control of Discrete Systems, York, United Kingdom*, 2013, pp. 157–162.

[10] Y. Koren, *The global manufacturing revolution.* Wiley, 2010.

[11] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. V. Brussel, "Reconfigurable manufacturing systems," *CIRP Annals – Manufacturing Technology*, vol. 48, pp. 527–540, 1999.

[12] R. Kumar, S. Takai, M. Fabian, and T. Ushio, "Maximally permissive mutually and globally nonblocking supervision with application to switching control," *Automatica*, vol. 41, no. 8, pp. 1299–1312, 2005.

[13] J. Li, X. Dai, and Z. Meng, "Automatic reconfiguration of petri net controllers for reconfigurable manufacturing systems with an improved net rewriting system-based approach," *Automation Science and Engineering, IEEE Transactions on*, vol. 6, no. 1, pp. 156–167, 2009.

[14] libFAUDES. (2006–2011) libFAUDES software library for discrete event systems. [Online]. Available: www.rt.eei.uni-erlangen.de/FGdes/faudes

[15] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren, "Reconfigurable manufacturing systems: Key to future manufacturing," *Journal of Intelligent Manufacturing*, vol. 11, pp. 403–419, 2000.

[16] A. Nooruldeen, "Supervisory control for reconfigurable manufacturing systems: Structural changes and re-usability of controllers," Master's thesis, Department of Electronic and Communication Engineering, Çankaya University, 2012.

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

[18] R. Sampath, H. Darabi, U. Buy, and L. Jing, "Control reconfiguration of discrete event systems with dynamic control specifications." *Automation Science and Engineering, IEEE Transactions on*, vol. 5, no. 1, pp. 84–100, 2008.

[19] K. Schmidt and C. Breindl, "Maximally permissive hierarchical control of decentralized discrete event systems," *Automatic Control, IEEE Transactions on*, vol. 56, no. 4, pp. 723–737, 2011.

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

[21] K. W. Schmidt, "Computation of supervisors for reconfigurable machine tools," Workshop on *Discrete Event Systems, Guadalajara, Mexico*, 2012, pp. 227–232.

[22] K. W. Schmidt, "Optimal configuration changes for reconfigurable manufacturing systems," IEEE Conference on *Decision and Control, Florence, Italy*, 2013.

[23] K. W. Schmidt, "Computation of supervisors for reconfigurable machine tools," *Journal of Discrete Event Dynamic Systems (accepted for publication)*, 2014.

[24] K. W. Schmidt and A. Nooruldeen, "State attraction under language specification for the reconfiguration of discrete event systems," *IEEE Transactions on Automatic Control (accepted for publication)*, 2014.

APPENDIX

CURRICCULUM VITAE

PERSONAL INFORMATION

Surname, Name: Hendi, Harith M. Khalid
Nationality: Iraqi (IRAQ)
Date and Place of Birth: 21 October 1985 , Baghdad
Marital Status: Single
Phone: +90 538 010 24 02/ +964 790 182 61 96
Email: harithhindi85@yahoo.com

EDUCATION

Degree	Institution	Year of Graduation
MS	Çankaya Univ. Electronic and Communication Engineering	2013/2014
BS	Univ. of Technology/Baghdad Control and Systems Engineering Department, Mechatronics branch	2006/2007
High School	Al-Mutamizean (Special Students)	2002/2003

FOREIGN LANGUAGES

Arabic, English, familiar with (French)

PUBLICATIONS

Harith M. Khalid, M. S. Kırık, and K. W. Schmidt, "Abstraction-based supervisory control for reconfigurable manufacturing systems" Workshop on *Dependable Control of Discrete Systems, York, United Kingdom*, 2013, pp. 157–162.

HOBBIES

Reading, Sport, Movies