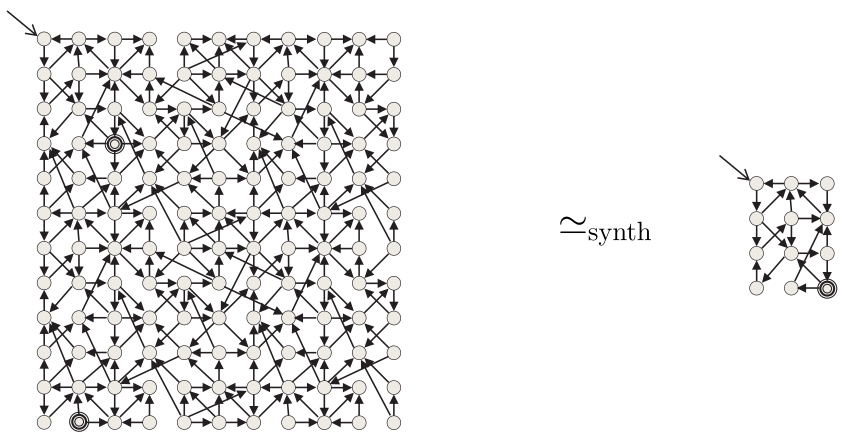

On Compositional Supervisor Synthesis for Discrete Event Systems

Sahar Mohajerani

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2012

Thesis for the degree of Licentiate of Engineering

On Compositional Supervisor Synthesis for Discrete Event Systems

by

Sahar Mohajerani

Department of Signals and Systems
Automation Group
Chalmers University of Technology
Göteborg, Sweden 2012

On Compositional Supervisor Synthesis for Discrete Event Systems
Sahar Mohajerani

This thesis has been prepared using LaTeX.

Technical Report No. R001/2012
School of Electrical Engineering
Chalmers University of Technology
ISSN 1403-266X

Department of Signals and Systems
Automation Group
Chalmers University of Technology
SE-412 96 Göteborg, Sweden

Phone: +46 (0)31 772 1000
E-mail: `mohajera@chalmers.se`

Printed by Chalmers Reproservice
Göteborg, Sweden, March 2012

To Ali & Marzieh
and the memory of Farideh

Abstract

Over the past decades, human dependability on technical devices has rapidly increased. Many activities of such devices can be described by sequences of events, where the occurrence of an event causes the system to go from one state to another. This is elegantly modeled by *automata*. Systems that are modeled in this way are referred to as *discrete event systems*. Many of these systems appear in settings that are safety critical, and small failures may result in huge financial and/or human losses. Having a control function is one way to guarantee system correctness.

Supervisory control theory, proposed by *Ramadge* and *Wonham*, provides a general framework to automatically calculate control functions for discrete event systems. Given a model of the system, the *plant*, to be controlled, and a *specification* of the desired behaviour, it is possible to automatically compute, i.e. *synthesise*, a *supervisor* that ensures that the specification is satisfied.

Usually, systems are *modular* and consist of several components interacting with each other. Calculating a supervisor for such a system in the standard way involves constructing the complete model of the considered system which may lead to the inherent complexity problem known as the *state-space explosion* problem. This problem occurs when composition of the components results in a model with a huge number of states, as the number of states grows exponentially with the number of components. This problem makes it intractable to examine the states of a system due to lack of memory and time.

This thesis uses a compositional approach to alleviate the state-space explosion problem. A compositional approach exploits the modular structure of a system to reduce the size of the model of the system. The thesis mainly focuses on developing the methodology for abstracting a system in a way that the final synthesis result is the same as it would have been for the non-abstracted system. The algorithms have been implemented in the discrete event system software tool Supremica and have been applied to compute modular supervisors for several large industrial models.

Keywords: Finite-state automata, abstraction, synthesis, supervisory control theory.

Acknowlegments

During the past two years of being Ph.D student, I have met many great people who in one way or another have helped me. I really like to mention everybody's name, but the space does not let me.

First of all my deepest gratitude goes to my supervisor Martin Fabian. Without your great support, encouragement and invaluable supervision this work would not be possible. You believed in me, even when I myself did not, and I can never thank you enough for that.

I also had the greatest pleasure to work with Robi Malik. You made the time I spent in New Zealand a wonderful time and that was the beginning of a very inspiring collaboration that has been continued for almost one year now. Thank you so much Robi for everything.

I am also grateful to Bengt Lennartson, the head of the automation group and all my colleagues and friends at the department specially people at the division of Automatic control, Automation and Mechatronics for all the cheerful memories and making the division a delightful place to work. Especially I would like to thank my two office roommates Zhennan and Patrik for all their help. Thanks to Sajed, Roozbeh, Mitra, Nina, Oskar and Mona for all the enjoyable "fikas". On the administrative side I would like to thank Madeleine Parsson, Agneta Kinnander and Lars Börjesson for being so helpful.

Finally I would like to thank my friends Azin, Arash, Sogol, Tohid, Laleh, Nima, Maryam and Aidin who always have helped me and cheered me up when I was down. Most importantly I would like to thank my family. Special thanks to my mother for her unlimited love and to my sisters and brother, you guys are amazing. And of course, I owe my heartfelt gratefulness to my love Ali. You were always there for me and supported me no matter what. From the bottom of my heart thank you!

Sahar Mohajerani
Göteborg, March 2012

List of appended papers

Paper I
Mohajerani, Sahar; Malik, Robi; Ware, Simon; Fabian, Martin: On the Use of Observation Equivalence in Synthesis Abstraction. *In Proceedings of the 3rd International Workshop on Dependable Control of Discrete Systems(DCDS'11)*, June 2011, Saarbrücken, Germany.

Paper II
Mohajerani, Sahar; Malik, Robi; Fabian, Martin: Nondeterminism Avoidance in Compositional Synthesis of Discrete Event Systems. *In Proceedings of IEEE Conference on Automation Science and Engineering 2011(CASE'11), pp. 19-24*, August 2011, Trieste, Italy.

Paper III
Mohajerani, Sahar; Malik, Robi; Fabian, Martin: Compositional Synthesis of Modular Supervisors Using Synthesis Equivalence. *Submitted to IEEE Transaction on Automatic Control, 2012.*

Other publications

The following papers are not included in this thesis due to overlap with the appended papers;

Paper IV
Mohajerani, Sahar; Malik, Robi; Ware, Simon; Fabian, Martin: THREE VARIATIONS OF OBSERVATION EQUIVALENCE PRESERVING SYNTHESIS ABSTRACTION . Göteborg : Chalmers University of Technology. (R - Department of Signals and Systems, Chalmers University of Technology;R008/2011)

Paper V
Mohajerani, Sahar; Malik, Robi; Ware, Simon; Fabian, Martin: Compositional Synthesis of Discrete Event Systems Using Synthesis Abstraction. *In Proceedings of the 23rd Chinese Control and Decision Conference*, May 2011, Mianyang, China.

Contents

Abstract i

Acknowledgments iii

List of publications v

Contents vii

Part I: Introductory Chapters xi

1 Introduction 1

1.1 Problem Statement 2
1.2 Main Contributions 3
1.3 Outline 3

2 Preliminaries 5

2.1 Modeling Formalism 5
2.1.1 Finite Automata 5
2.1.2 Events and Languages 8
2.2 Equivalence Relation 8

3 Supervisory Control Theory 11

3.1 Requirements for Supervisors 12
3.1.1 Nonblocking 12
3.1.2 Controllability 12
3.1.3 Least Restrictiveness 13
3.2 Synthesis 13
3.2.1 Synthesis Algorithm 13

3.3 Translation of Specifications to Plants 14
3.3.1 Translated Specifications vs Forbidden States 16
3.3.2 Need for Transforming Specifications to Plants 16
3.3.3 Modular Approach and Forbidden States 17

4 Compositional Synthesis 21
4.1 General Compositional Approach 22
4.1.1 Local Events 22
4.2 Equivalence Relations 24
4.2.1 Supervision Equivalence 24
4.2.2 Synthesis Abstraction 25
4.2.3 Renaming and Synthesis Equivalence 27
4.3 Abstraction Methods Preserving Synthesis Equivalence 29
4.3.1 Observation Equivalence 31
4.3.2 Selfloop Removal 35
4.3.3 Halfway Synthesis 36
4.3.4 Partial Supervisors 37
4.4 Experimental Results 37
4.4.1 Test Examples 38
4.4.2 Heuristics 39
4.4.3 Results 39

5 Summary of Included Papers 43

6 Concluding Remarks and Future Work 45

References 47

Part II: Publications 51

Paper I: On the Use of Observation Equivalence in Synthesis Abstraction 53
Abstract 55
1 Introduction 55
2 Preliminaries and Notation 56
2.1 Events and Languages 56
2.2 Nondeterministic Automata 57
2.3 Supervisory Control Theory 58
3 Compositional Synthesis 60
3.1 General Compositional Approach 60
3.2 Synthesis Abstraction 60

4 Methods of Abstraction 61
4.1 Observation Equivalence 61
4.2 Bisimulation 62
4.3 Uncontrollable Observation Equivalence 63
4.4 Synthesis Observation Equivalence 64
4.5 Relationship to Projection 65
5 Example 67
6 Conclusions 69
References 69

Paper II: Nondeterminism Avoidance in Compositional Synthesis of Discrete Event Systems **73**
Abstract 75
1 Introduction 75
2 Preliminaries and Notation 76
2.1 Events and Languages 76
2.2 Nondeterministic Automata 77
2.3 Supervisory Control Theory 78
3 Renaming in Compositional Synthesis 79
3.1 General Compositional Approach 80
3.2 Renaming 81
3.3 Renaming and Compositional Synthesis 83
3.4 Abstractions Preserving Synthesis Equivalence 84
4 Example 86
5 Conclusions 89
References 89

Paper III: Compositional Synthesis of Modular Supervisors Using Synthesis Equivalence **93**
Abstract 95
1 Introduction 95
2 Preliminaries 97
2.1 Events and Languages 97
2.2 Finite-State Automata 97
2.3 Supervisory Control Theory 98
3 Compositional Synthesis 101
4 Synthesis Triple Rewrite Operations 106
4.1 Synchronous Composition, Renaming, Selfloop Removal 106
4.2 Observation Equivalence 107
4.3 Halfway Synthesis 111
5 Experimental Results 112

6 Conclusions 116
References 117
Appendices 122
A Proofs for Renaming and Selfloop Removal 122
B Proof of Synthesis Observation Equivalence 125
C Proof for Halfway Synthesis 130

Part I
Introductory Chapters

Chapter 1

Introduction

The modern human being is a hybrid of a traditional homo-sapiens with fancy electronic gadgets. We use a electronic devices everyday, and it seems we are never more than a meter away from our cellphones. These devices are designed to help us live our lives easier, and one of our most important requirements on them is consistency. We expect the devices to work in a certain way when we provide them with a certain input. In engineering terms, everything between the input that we provide and the output we see is broadly termed a *system*. The coffee machine, the printer and industrial robots are some examples of systems.

When dealing with different systems, many questions about the properties of systems arises. For example in the case of a mobile phone one may wonder: what will happen if I push a specific button? For a nuclear plant a question could be: What will happen if a nuclear reactor core becomes too hot? Experimentation is one way to answer these kind of questions. In many cases, experiments are very expensive or could even be dangerous. An alternative to answer such questions is to *model* the systems behaviour.

Modeling can be viewed in two perspectives. In the first point of view, using physical knowledge, mathematical equations that describe the output of a system given an input is derived. Newton's law, gravity laws and differential equations are some tools used in this context. The other way of viewing system modeling is to describe a system behaviour by sequences of events, where the occurrence of an event causes the system to go from one state to another. The tools to model the behaviour of such systems are *events* and *states*. For example, when a coffee machine is out of coffee beans, it goes from a working state to an idle state, and becoming out of beans is the event. Such system models are referred to as *discrete event systems* and are the main focus of this thesis. Discrete event systems may vary from very simple household devices like a simple coffee machine or cooking devices, to more sophisticated and complicated systems including aircraft electronics, industrial and manufacturing systems.

1.1 Problem Statement

Consider a coffee machine that fills your glass with tea even though you asked for coffee. In this case you may just accept the tea and get back to work. However, many applications of discreet event systems take place in settings that are safety critical and small failures may result in huge financial and/or human losses. Having a control function is one way to guarantee system correctness.

In 1989, *Ramadge* and *Wonham* [29] proposed a framework to calculate a controlling agent, called a *supervisor*, for discrete event systems. This framework is called the *supervisory control theory.* Given a model of a system to be controlled, the *plant*, and the desired behaviour of the controlled system, the supervisory control theory proposes methods to design and automatically construct a *supervisor* in such a way that the controlled system of plant and supervisor always acts as desired.

For simple systems consisting of a small number of states, calculating the supervisor can be done straightforwardly. However, nowadays the systems are becoming more and more complex and each system consists of several interacting subsystems. Such systems are referred to as *modular systems.* Using the standard approach to calculate the supervisor for such systems involves explicitly representing the entire system by a single model which may consist of millions of states. This inherent complexity problem is known as the *state-space explosion* problem. A brute force approach to calculate a supervisor is to go through all states of a system and remove undesirable states. The state space explosion makes it intractable to analyse all states of a system due to lack of memory and time.

The state-space explosion problem could occur when one tries to model a modular system by a single representation. However, it is possible to use the knowledge of modularity of the system to our advantage. One way to exploit the modularity of systems is to use *compositional approaches.* To avoid the state-space explosion problem, a compositional approach tries to build a single representation of a system in an incremental way. The general approach is as follows. First the subsystems are simplified by merging related states until no further simplification (also called *abstraction*) is possible, and then the subsystems are combined together one by one and simplified again in each iteration. This process is repeated until it results in one final relatively simple model. This simple model is finally used for synthesis.

1.2 Main Contributions

The main focus of this thesis is to use the compositional approach for calculating a supervisor. Questions that immediately arise are:

- What is the property that should be preserved after simplification?
- Is it possible to find methods in order to simplify subsystems efficiently?
- Does the compositional approach make supervisor calculation efficient?

Attempting to answer these questions results in the following contributions in this thesis:

- In this work, maintaining the same closed loop behaviour is the property to be preserved after simplification. Also the computed supervisor in this thesis is modular in that it consists of several interacting components.
- The main focus of this thesis is to develop abstraction methods in the compositional approach such that the final closed loop behaviour is the same as it would have been for the non-abstracted system. The abstraction methods presented are mostly based on a well known abstraction method called *observation equivalence* and it is shown how observation equivalence can be strengthened to be applicable in the compositional synthesis framework.
- The algorithms proposed in this work have been implemented in the DES software tool Supremica and have been applied to compute supervisors for several benchmark examples. The experimental results show that the method successfully computes modular supervisors for a set of very large industrial models.

1.3 Outline

The first two chapters, Chapter 2 and Chapter 3, give the preliminaries and the background of supervisory control theory. In Chapter 4 the compositional synthesis proposed in this work is described. This chapter also presents the different abstraction methods to reduce complexity of systems and several benchmark examples. A summary of the appended papers is provided in Chapter 5. Finally some concluding remark are given in Chapter 6.

Chapter 2

Preliminaries

Many activities of technical devices in our daily uses can be described by sequences of events. These systems are referred to as *discrete event systems* (DES). A DES is a dynamic system with events and states as its basic elements. Events represent incidents that cause transitions from one state to another and states describe the current system status after occurrence of an event. Such systems vary from simple household devices to complicated systems such as aircraft electronics, industrial and manufacturing systems.

2.1 Modeling Formalism

A prerequisite to analyse discrete event systems is developing suitable models that can accurately represent the activities of the system. Different modeling formalisms have been used in the literature to represent discrete event systems, for instance, automata [3], petri nets [16] and formal languages [8, 29]. In this thesis *finite automata* are used to represent the behaviour of discrete events systems.

2.1.1 Finite Automata

Discrete event systems behaviour are typically modeled by deterministic automata, but in our case nondeterministic automata may arise as intermediate results during abstraction.

Definition 1 *A finite-state automaton is a tuple $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$, where*

- *Σ is the* alphabet *which is a finite set of events,*
- *Q is the finite set of* states*,*

- $\rightarrow \subseteq Q \times \Sigma \times Q$ *is the* state transition relation*,*
- $Q^\circ \subseteq Q$ *is the set of* initial states.

The following notation and convention are used in this thesis.

- The transition relation is written in infix notation $x \xrightarrow{\sigma} y$, and is extended to traces in Σ^* by letting $x \xrightarrow{\varepsilon} x$ for all $x \in Q$ where ε is the empty trace, and $x \xrightarrow{s\sigma} z$ if $x \xrightarrow{s} y$ and $y \xrightarrow{\sigma} z$ for some $y \in Q$.
- $x \xrightarrow{s}$ means that $x \xrightarrow{s} y$ for some $y \in Q$, and $x \rightarrow y$ means that $x \xrightarrow{s} y$ for some $s \in \Sigma^*$. These notations also apply to state sets, $X \xrightarrow{s}$ for $X \subseteq Q$ means that $x \xrightarrow{s}$ for some $x \in X$, and to automata, $G \xrightarrow{s}$ means that $Q^\circ \xrightarrow{s}$.
- A special *termination event*, ω, is used to denote marking of a state. A special requirement is that states reached by the termination event, ω, do not have any outgoing transitions, i.e., if $x \xrightarrow{\omega} y$ then there does not exist $\sigma \in \Sigma$ such that $y \xrightarrow{\sigma}$. This ensures that the termination event, if it occurs, is always the final event of any trace. The traditional set of marked states is $Q^\omega = \{\, x \in Q \mid x \xrightarrow{\omega} \,\}$ in this notation. For graphical simplicity, states in Q^ω are shown shaded in the figures of this thesis, to avoid explicitly showing ω-transitions.

Automaton G is *deterministic*, if $|Q^\circ| \leq 1$, and $x \xrightarrow{\sigma} y_1$ and $x \xrightarrow{\sigma} y_2$ always implies $y_1 = y_2$.

Some automata are structurally related. One such relation is when the structure of one automaton is contained within another, and they both have the same alphabet. This structural relation is important for the algorithm to be described.

Definition 2 *Let* $G_1 = \langle \Sigma_1, Q_1, \rightarrow_1, Q_1^\circ \rangle$ *and* $G_2 = \langle \Sigma_2, Q_2, \rightarrow_2, Q_2^\circ \rangle$*, where* $\Sigma_1 = \Sigma_2$*, be two automata.* G_1 *is a subautomaton of* G_2*, written* $G_1 \subseteq G_2$*, if* $Q_1 \subseteq Q_2$*,* $\rightarrow_1 \subseteq \rightarrow_2$*, and* $Q_1^\circ \subseteq Q_2^\circ$*.*

It can be of interest to *restrict* the behaviour of an automaton to a subset of its states. Restriction is important when we talk about *supervisory control theory* in Chapter 3, where a system behaviour is restricted to a desired behaviour.

Definition 3 *The* restriction *of* $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ *to* $X \subseteq Q$ *is*

$$G_{|X} = \langle \Sigma, X, \rightarrow_{|X}, Q^\circ \cap X \rangle \ , \tag{2.1}$$

where $\rightarrow_{|X} = \{\, (x, \sigma, y) \in \rightarrow \mid x, y \in X \,\}$*.*

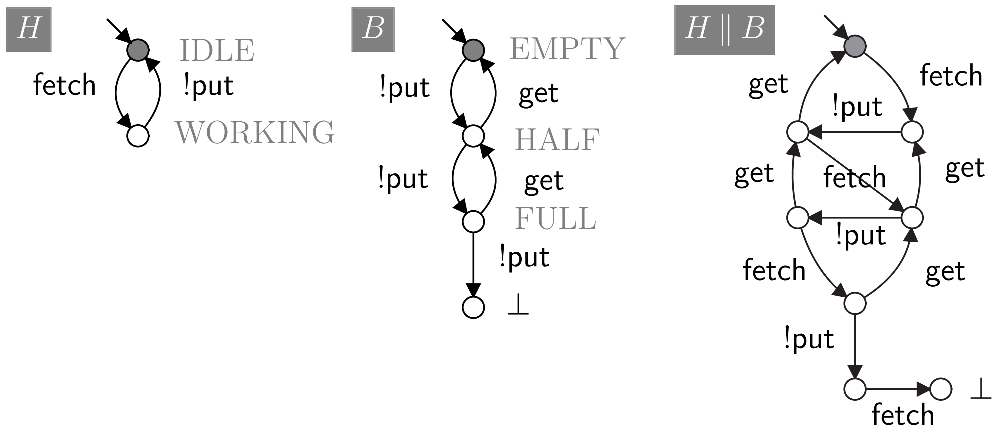

Figure 2.1: Automata for manufacturing system, used in Example 1.

Most discrete event systems consist of several subsystems running in parallel. When these components are brought together to interact, synchronisation in the style of [19] is used.

Definition 4 *Let* $G_1 = \langle \Sigma_1, Q_1, \rightarrow_1, Q_1^\circ \rangle$ *and* $G_2 = \langle \Sigma_2, Q_2, \rightarrow_2, Q_2^\circ \rangle$ *be two automata. The* synchronous composition *of* G_1 *and* G_2 *is defined as*

$$G_1 \parallel G_2 = \langle \Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \rightarrow, Q_1^\circ \times Q_2^\circ \rangle \tag{2.2}$$

where

$$\begin{cases} (x,y) \xrightarrow{\sigma} (x',y') & \textit{if } \ \sigma \in \Sigma_1 \cap \Sigma_2,\ x \xrightarrow{\sigma}_1 x',\ y \xrightarrow{\sigma}_2 y'\,; \\ (x,y) \xrightarrow{\sigma} (x',y) & \textit{if } \ \sigma \in \Sigma_1 \setminus \Sigma_2,\ x \xrightarrow{\sigma}_1 x'\,; \\ (x,y) \xrightarrow{\sigma} (x,y') & \textit{if } \ \sigma \in \Sigma_2 \setminus \Sigma_1,\ y \xrightarrow{\sigma}_2 y'\,. \end{cases}$$

In synchronous composition, an event shared between two automata can be executed only if it is executed by the two automata simultaneously. However there is no such constraint on the non-shared *local* events.

Example 1 *Figure 2.1 shows automata models of a simple manufacturing system consisting of a handler* H *and a buffer* B *with capacity two. The handler fetches a workpiece (*fetch*) and then puts it into the buffer (*!put*), and afterwards the buffer releases the workpiece (*get*). The behaviour of the handler is modeled by automaton* H *in Figure 2.1. The behaviour of the buffer is given by the automaton* B*. The buffer can store only two workpieces, adding more workpieces causes overflow, represented by the state* $\perp$*. Figure 2.1 also shows the synchronised model* $H \parallel B$ *which consists of 8 states.*

2.1.2 Events and Languages

A behaviour of a DES is described over a finite set of events Σ. Σ^* is the set of all finite traces of events from Σ, including the *empty trace* ε. A subset $L \subseteq \Sigma^*$ is called a *language*. The language of an automaton is the set of all traces generated by the automaton.

Definition 5 *Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton. The language of G, denoted $\mathcal{L}(G)$, is defined as*

$$\mathcal{L}(G) = \{\, s \in \Sigma^* \mid G \xrightarrow{s} \,\} \tag{2.3}$$

The following notations and definitions are considered for languages. The concatenation of two traces $s, t \in \Sigma^*$ is written as st. A trace $s \in \Sigma^*$ is called a *prefix* of $t \in \Sigma^*$, written $s \sqsubseteq t$, if $t = su$ for some $u \in \Sigma^*$. For $\Omega \subseteq \Sigma$, the *natural projection* $P_\Omega \colon \Sigma^* \rightarrow \Omega^*$ is the operation that removes from traces $s \in \Sigma^*$ all events not in Ω [9]. The projection P_Ω can be extended to languages, by applying it to all the strings in the language.

Example 2 *Let $\Sigma = \{\alpha, \beta\}$ and consider subset $\Omega = \{\alpha\}$. Let $\mathcal{L} = \{\varepsilon,\ \alpha,\ \alpha\beta\}$. Then $P_\Omega(\mathcal{L}) = \{\varepsilon, \alpha\}$.*

To describe discrete event systems behaviour regular languages are commonly used in the literature [8, 29]. However, regular languages can equivalently be depicted by automata. In this thesis, automata are used to model a DES since it is not straightforward to show nondeterminism by language and the main focus of this work is on abstraction which may cause nondeterminism. Furthermore, algorithms typically work on automata although languages are used for modeling.

2.2 Equivalence Relation

An *equivalence relation* is a binary relation that partitions a set into disjoint subsets. In this thesis, equivalence relation $\sim$ is applied to state set Q of automata. The equivalence class of state $x \in Q$ with respect to the equivalence relation $\sim$, written as $[x]$, is $[x] = \{\, x' \in Q \mid x \sim x' \,\}$ and $Q/{\sim} = \{\, [x] \mid x \in Q \,\}$ is the set of all equivalence classes modulo $\sim$.

Since an equivalence relation $\sim$ is reflexive, symmetric and transitive the following holds respectively,

- $x \sim x$ for all $x \in Q$,
- if $x_1 \sim x_2$ then $x_2 \sim x_1$ and $x_1, x_2 \in [x_1] = [x_2]$

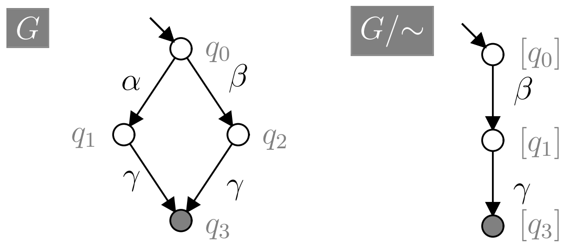

Figure 2.2: Automata of example 3.

- if $x_1 \sim x_2$ and $x_2 \sim x_3$ then $x_1 \sim x_3$ and $x_1, x_2, x_3 \in [x_1] = [x_2] = [x_3]$.

Applying an equivalence relation on the state set of automaton results in the quotient automaton.

Definition 6 *Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton and let $\sim \subseteq Q \times Q$ be an equivalence relation. The* quotient automaton *of G modulo $\sim$ is*

$$G/\sim = \langle \Sigma, Q/\sim, \rightarrow/\sim, Q^\circ/\sim \rangle \ , \tag{2.4}$$

where $\rightarrow/\sim = \{ [x] \xrightarrow{\sigma} [y] \mid x \xrightarrow{\sigma} y \}$.

The quotient automaton is an abstracted automaton which can be obtained by merging the states of the equivalence classes or, equivalently, regard the equivalence classes as the states of the abstracted automaton.

Example 3 *Consider the automaton $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ in Figure 2.2. Let $\sim \subseteq Q \times Q$ be an equivalence relation such that $q_0 \in [q_0]$, $q_1, q_2 \in [q_1]$ and $q_3 \in [q_3]$. Figure 2.2 also shows $G/\sim$ which is the quotient automaton of G and is obtained by merging the states of the equivalence classes.*

Chapter 3

Supervisory Control Theory

A discrete event system usually consists of sets of plant components modeled by automata. Plant automata can be seen as event generators and describe the behaviour of the uncontrolled system. Usually, the system behaviour is not acceptable in that it violates some safety or nonblocking constraint. A specification describes the desired behaviour and is also modeled by automata. To avoid violation of the specification by the uncontrolled plant, a supervisor needs to be calculated.

Given a *plant* automaton G and a *specification* automaton K, the *supervisory control theory* [29] provides a method to synthesise a supervisor S that restricts the behaviour of the plant such that the specification is always fulfilled.

Figure 3.1, shows the feedback loop of supervisor and plant. The plant generates events and the supervisor, based on the generated events (as in Figure 3.1), decides whether to enable or disable the currently possible plant events. Thus, the supervisor is itself incapable of generating events and only enables or disables them.

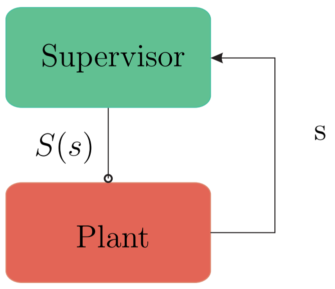

Figure 3.1: The feedback loop of supervisor and plant.

3.1 Requirements for Supervisors

A plant describes everything that the uncontrolled system is capable of doing and the specification expresses the desired behaviour. A supervisor is designed to restrict the plant behaviour such that the plant does not violate the specification. Besides this essential requirement, there are more requirements that a supervisor should fulfill.

3.1.1 Nonblocking

In automata, *marked states* are used to represent completion of a (sub-)task. It is desirable for a system to be able to complete tasks or in other words to be free from *deadlock* and *livelock*. Formally speaking, deadlock refers to a situation when a system is in an unmarked state from which there is no outgoing transition, such as, two or more systems waiting for each other to release a common resource. Livelock is similar to deadlock, except that the system is not blocked but it is in a loop that it can not get out from.

One crucial issue to consider while computing a supervisor is that the controlled system can always complete at least some tasks. Such a supervisor is referred to as a *nonblocking* supervisor.

Definition 7 *Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$. A state $x \in Q$ is called* reachable *in G if $G \rightarrow x$, and* coreachable *if $x \xrightarrow{s\omega}$ for some $s \in \Sigma^*$. G is called* nonblocking *if every reachable state is coreachable.*

Given a plant G and a supervisor S the resultant closed loop behaviour is $G \parallel S$, and of course this automaton should be nonblocking.

3.1.2 Controllability

For the purpose of supervisory control, the alphabet of a system is partitioned into two disjoint subsets, the set Σ_c of *controllable* events and the set Σ_u of *uncontrollable* events. Controllable events can be disabled by a supervisor but uncontrollable events can not.

Considering uncontrollable events, one requirement for the computed supervisor is that it never tries to disable an executable uncontrollable event in order to restrict the system.

Definition 8 *Let $G = \langle \Sigma_G, Q_G, \rightarrow_G, Q_G^\circ \rangle$ and $K = \langle \Sigma_K, Q_K, \rightarrow_K, Q_K^\circ \rangle$ be two automata. K is* controllable *with respect to G if, for every trace $s \in (\Sigma_G \cup \Sigma_K)^*$, every state x of K, and every uncontrollable event $\upsilon \in (\Sigma_G \cap \Sigma_K \cap \Sigma_u)$ such that $K \xrightarrow{P_{\Sigma_K}(s)} x$ and $G \xrightarrow{P_{\Sigma_G}(s)\upsilon}$, it holds that $x \xrightarrow{\upsilon}$ in K.*

This definition says that given a plant G and a supervisor S, the supervisor S is controllable with respect to plant G if the occurrence of an uncontrollable event does not lead to a string which is not acceptable by the supervisor.

3.1.3 Least Restrictiveness

The purpose of a supervisor is to restrict a plant behaviour to fulfill a specification. Typically, there is no unique controllable and nonblocking supervisor. For instance the null automaton could be a supervisor and since it disables all the events generated by the plant it is a nonblocking and controllable supervisor that fulfills any specification. However, the null automaton is not a useful supervisor and it is desirable to restrict the plant behaviour as little as possible. Such a supervisor is referred to as the *least restrictive* supervisor. In this thesis, the term “supervisor” refers to a least restrictive, controllable and nonblocking supervisor and this supervisor is unique [29].

3.2 Synthesis

If a system does not satisfy the specification, the question arises whether it is possible to remove states that violate the specification. This question is answered by fundamental results in [29], where it is shown that for every given regular language, there exists a unique maximal sublanguage that is controllable, nonblocking and least restrictive.

3.2.1 Synthesis Algorithm

In this thesis, the system behaviour is modeled by automata. To cope with automata based modeling, the result in [29] can be reformulated in automata form, using an iteration on the automaton state set [26]. The following algorithm will calculate the largest set of controllable and coreachable states of a given automaton.

Given the automaton G, the algorithm iteratively identifies and removes blocking and uncontrollable states and it returns the maximal set of controllable and nonblocking states. The largest controllable and nonblocking *subautomaton* of G is obtained by restricting G to this set which is written $\sup\mathcal{CN}_\Gamma(G)$ [26]. When $\Gamma = \Sigma_u$, the subscript is omitted, i.e., $\sup\mathcal{CN}(G) = \sup\mathcal{CN}_{\Sigma_u}(G)$.

In the finite-state case, the iteration is guaranteed to converge, and the complexity is $O(|Q||{\rightarrow}|)$, where $|Q|$ and $|{\rightarrow}|$ are the numbers of states and

Algorithm 1 Maximal state set $\Theta_{G,\Gamma}(Q)$

PRECONDITION : A given automaton $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ and set of uncontrollable events $\Gamma \subseteq \Sigma_u \subseteq \Sigma$.

POSTCONDITION : A set $\Theta \subseteq Q$, which is the largest set of controllable and coreachable states of G.

```
$i \leftarrow 0$
$\Theta^0 \leftarrow Q$
**repeat**
$\Theta^{i+1} \leftarrow \Theta^i$
$\forall x \in \Theta^{i+1}$ **if** $\nexists t$ such that $x \xrightarrow{t\omega}$ **then**
    $\Theta^{i+1} \leftarrow \Theta^{i+1} \setminus x$
**end if**
$\forall x \in \Theta^{i+1}$ **if** $x \xrightarrow{\upsilon} y$ and $y \notin \Theta^{i+1}$ for some $\upsilon \in \Gamma$ **then**
    $\Theta^{i+1} \leftarrow \Theta^{i+1} \setminus x$
**end if**
**until** $\Theta^{i+1} = \Theta^i$
**return** $\Theta^{i+1}$
```

transitions of the automaton. The size of Q and $\rightarrow$ grows exponentially with the number of components, and [17] shows that the compositional synthesis problem is NP-complete.

3.3 Translation of Specifications to Plants

The synthesis operation described in Section 3.2 only performs synthesis for a plant automaton G. In order to apply this synthesis algorithm to control problems that also involve specifications, the transformation proposed in [15] can be used. A specification automaton is transformed into a plant by adding, for every uncontrollable event that is not enabled in a state, a transition to a new blocking state $\perp$. This essentially transforms all initial controllability problems into blocking problems.

Definition 9 *[15] Let $K = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a specification. The* complete plant automaton *$K^\perp$ for K is*

$$K^\perp = \langle \Sigma, Q \cup \{\perp\}, \rightarrow^\perp, Q^\circ \rangle \tag{3.1}$$

where $\perp \notin Q$ is a new state and

$$\rightarrow^\perp = \rightarrow \cup \{ (x, \upsilon, \perp) \mid x \in Q, \upsilon \in \Sigma_u, x \overset{\upsilon}{\not\rightarrow} \} . \tag{3.2}$$

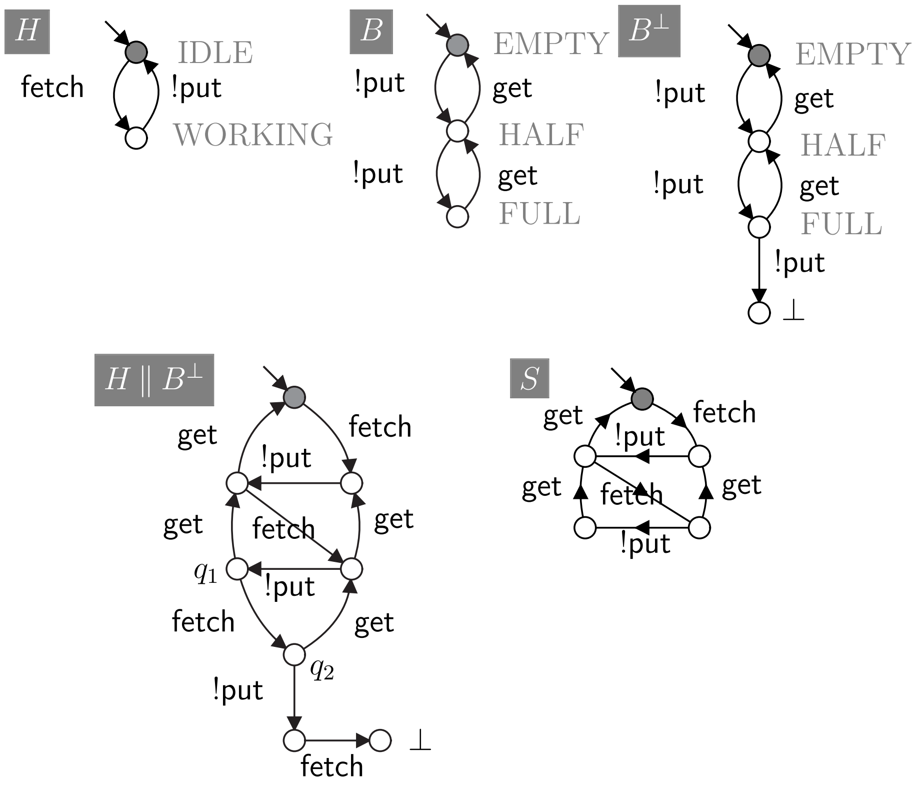

Figure 3.2: Automata of Example 4.

Given a plant G, synthesising the least restrictive nonblocking and controllable behaviour that is allowed by specification K involves synchronising the plant with the translated specification and computing $\sup\mathcal{CN}(G \parallel K^\perp)$ [15]. This way of calculating a supervisor is referred to as the *monolithic* approach, and the supervisor obtained is a *monolithic* supervisor which consists of a single automaton. However, calculating a supervisor using the monolithic approach is impeded by state space explosion problem. Since usually, a system behaviour is modeled as a set of automata it is possible to exploit the modular structure of the system and design a *modular* supervisor.

Example 4 *Figure 3.2 shows a model of a simple manufacturing system consisting of a handler (plant H) and a buffer (specification B). The handler fetches a workpiece (*fetch*) and puts it into the buffer (*!put*), afterwards the buffer releases it (*get*). Events* fetch *and* get *are controllable, while* !put *is uncontrollable. $B^\perp$ is the complete plant automaton for B. This system is blocking, since the trace* fetch !put fetch !put fetch !put fetch *takes $B^\perp$ to state $\perp$, representing the undesired possibility of buffer overflow. To prevent this, event* !put *needs to be disabled in state q_2 of $H \parallel B^\perp$. How-*

ever, !put *is an uncontrollable event that can not be disabled by the supervisor, so the least restrictive solution is obtained by disabling the controllable event* fetch *in state* q_1*. Figure 3.2 shows the resultant least restrictive supervisor* $S = \sup\mathcal{CN}(H \parallel B^{\perp})$*.*

3.3.1 Translated Specifications vs Forbidden States

The idea of translating the specification is proposed in [15] and has been implemented in the DES software tool *Supremica* [2]. However, typical synthesis algorithm [1,21,29] does not use translated specification and it can be interesting to briefly compare the two approaches.

In the original algorithm first the plants and the specifications are synchronised. During the synchronisation process, whenever an uncontrollable event in the plant is disabled by the specification, the synchronised state is marked as *forbidden*. The set of forbidden states is iteratively extended by including blocking states and states that have an uncontrollable outgoing transition to a forbidden state. To calculate the supervisor all the forbidden states are removed. It has been shown in [15] that the same supervisor is obtained by both approaches.

Example 5 *Consider again the model of a simple manufacturing system consisting of a handler (plant H) and a buffer (specification B) which is shown in Figure 3.3. State* q_5 *in* $H \parallel B$ *is an uncontrollable state that represents buffer overflow. We can note that this state is equivalent to the state* q_2 *in Figure 3.2 which uncontrollably takes the systems to* $\perp$*. This state is a forbidden state and crossed out in Figure 3.3. To prevent the system to reach the forbidden state* q_5*, event* fetch *needs to be disabled. Fig 3.3 shows* S *which is the least restrictive controllable and nonblocking supervisor and it is equal to the supervisor calculated in Example 4.*

3.3.2 Need for Transforming Specifications to Plants

In Supremica's original synthesis algorithm, uncontrollable states are marked as forbidden and synchronisation of a forbidden state with a non-forbidden state results in a forbidden state. Calculating the supervisor in the monolithic way involves synchronising the plants and the specifications once, which makes it possible to identify the forbidden states, and removing them using the original synthesis algorithm. However, there are algorithms, for example the *compositional algorithm* described in Chapter 4, that synchronise plants and specifications step by step in an incremental way and using forbidden states may result in a supervisor that is not least restrictive.

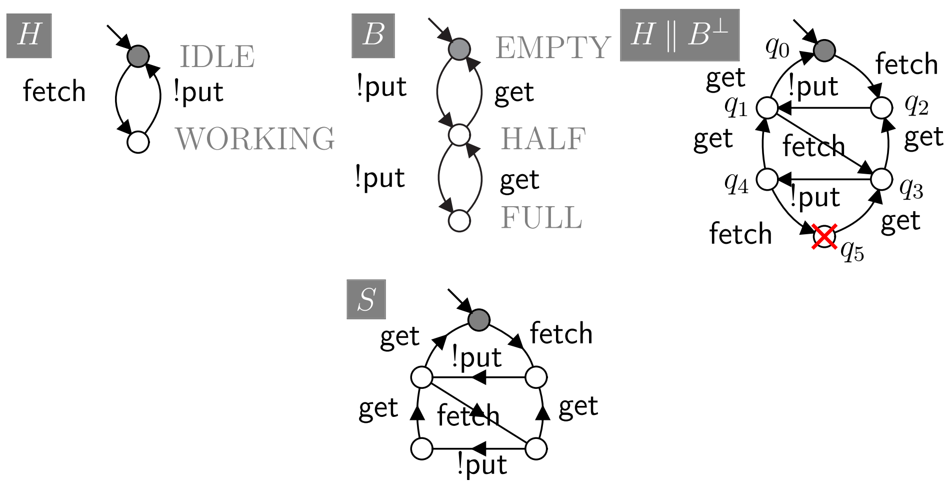

Figure 3.3: Automata of Example 5.

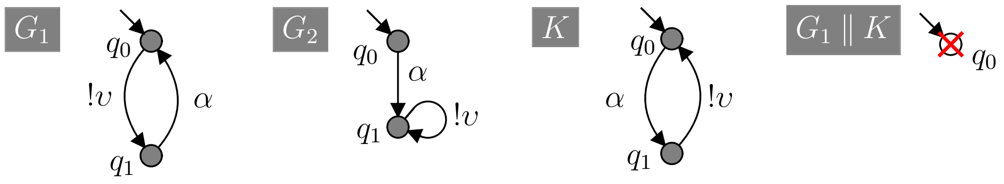

Figure 3.4: Automata of Example 6. Calculating the supervisor using the compositional approach without transforming K to plant.

Example 6 *Consider the modular plant $G = \{G_1, G_2\}$ and the specification K shown in Figure 3.4. Using the compositional approach, first G_1 is synchronised by K. Since $G_1 \xrightarrow{\upsilon} q_1$ and K does not have such a transition, the state q_0 in $G_1 \| K$ becomes a forbidden state. Synchronising $G_1 \| K$ by G_2 results in a single forbidden state. Synthesising a single forbidden state results in the null automaton. Figure 3.5 shows $K^\perp$ and $G_1 \| K^\perp$. Now synchronisation of $G_1 \| K^\perp$ and G_2 results in a single marked state and the synthesis result is a single marked state which is the least restrictive, controllable and nonblocking supervisor.*

3.3.3 Modular Approach and Forbidden States

The supervisor that is calculated by the algorithm in Section 3.2.1 is the least restrictive, controllable and nonblocking supervisor. Calculating such a supervisor for models with huge number of states could be intractable. To avoid such a problem, different approaches have been proposed. The *modular*

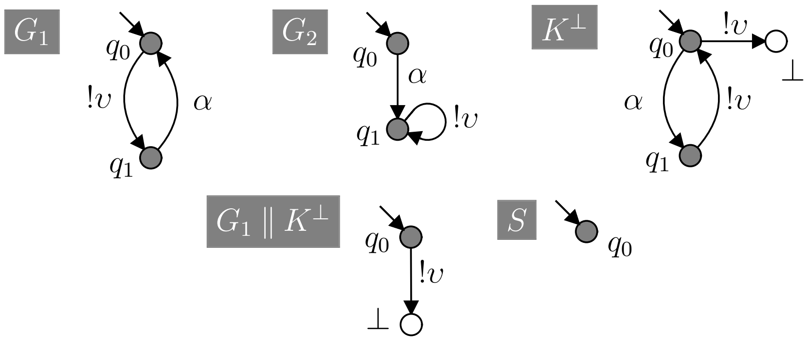

Figure 3.5: Automata of Example 6. Translating the specification to plant results in the least restrictive supervisor in the compositional approach.

approach can be mentioned as one such approach [1,6,11,34]. The calculated supervisor is a least restrictive and controllable modular supervisor. This approach can be described by the following algorithm.

The modular approach uses forbidden states to mark uncontrollable and blocking states. This seems to contradict the result in Section 3.3.2 where a conclusion has been drawn that using forbidden states for algorithms that calculate the supervisor in an incremental way results in a non-least restrictive supervisor. However, in Example 6 a non-least restrictive supervisor was obtained because not all the plants sharing the uncontrollable event with the specification are considered when forbidden states were identified.

To guarantee the least restrictive supervisor, all plants sharing uncontrollable events with a specification K must be included in the synthesis plus all plants sharing uncontrollable events with the originally picked plants. This is what happens in steps 6-15 in Algorithm 2. Note that this may mean that the modular approach in the worst case synthesises a supervisor for the entire monolithic plant. In practice, though, the modular approach seems to work rather well for most real industrial systems.

Example 7 *Consider again the modular system $G = \{G_1, G_2\}$ and the specification K shown in Figure 3.4. To use the modular approach, first K is picked and since G_1 and G_2 share uncontrollable events with K they are also picked. Calculating a supervisor for K, G_1 and G_2 results in a least restrictive supervisor.*

Step 10-14 in Algorithm 2 can be done more efficiently as the plants can be added incrementally when an uncontrollable event not in the specification is encountered during synthesis.

Though the modular approach of Supremica generates for each specification a supervisor, S_i for $1 = 1, \cdots, m$, that is least restrictive, con-

Algorithm 2 Modular Approach

PRECONDITION : A given a set of plants $G = \{G_1, \cdots, G_n\}$ and a set of specifications $K = \{K_1, \cdots, K_m\}$.

POSTCONDITION : A set S, which is a controllable and least restrictive supervisor.

```
1: $S \leftarrow \emptyset$
2: $P \leftarrow \emptyset$
3: $l \leftarrow 1$
4: repeat
5: remove $K_l$ from $K$ and add it to $P$
6: for $i = 1, \cdots, n$ do
7:    if $\Sigma_{G_i} \cap \Sigma_{K_l} \cap \Sigma_u \neq \emptyset$ then
8:       add a copy of $G_i$ to $P$
9:    end if
10:   for $j = 1, \cdots, n$ do
11:      if $j \neq i$ and $\Sigma_{G_j} \cap \Sigma_P \cap \Sigma_u \neq \emptyset$ then
12:         add a copy of $G_j$ to $P$
13:      end if
14:   end for
15: end for
16: calculate sup$\mathcal{CN}(\|P)$ and add it in $S$
17: $l \leftarrow l + 1$
18: until $l = m$
19: return $S$
```

trollable and nonblocking, this does not guarantee the modular supervisor $S = S_1 \parallel \cdots \parallel S_m$ is globally nonblocking. The compositional verification proposed in [14] can be used to verify whether the controlled system is blocking or not. However, [14] gives no clues on how to handle the case when the modular system is blocking.

To achieve the least restrictive supervisor with the modular approach, all plants sharing uncontrollable events, directly or indirectly, with a specification must be considered. This limits the degree of freedom when choosing the automata to compose. As shown, this limitation can be overcome by transforming specification to plants.

Chapter 4

Compositional Synthesis

Usually discrete event systems are modular in that the model of the system consists of a set of plant components and a set of specifications, all interacting with each other. Calculating a supervisor for a modular system in the standard way involves building an explicit monolithic model which may lead to the inherent complexity problem known as the *state-space explosion* problem. This problem occurs when synchronisation of the components results in an automaton with a huge number of states, as the number of states grows exponentially with the number of components. This problem makes it intractable to examine the global states of a system due to lack of memory and time. Consequently, constructing the monolithic model of the system is not efficient and methods to exploit the modular structure are needed. Such a method used in this thesis is the *compositional approach*. The compositional approach has been successfully used for *verification* of discrete event systems [4,10,14,32,33]. However, we are concerned with more than giving a "yes" or "no" answer and the task is to iteratively remove states that violate the specification.

In this chapter, first the general compositional approach is described. The general compositional approach needs a proper notion of equivalence in order to be used for synthesis analysis. Section 4.2 describes and compares different equivalence relations that have been used for compositional synthesis. Then, Section 4.3 presents different ways of reducing the size of subsystems, preserving the equivalence relation that is used, and finally Section 4.4 applies the method to several benchmark examples.

4.1 General Compositional Approach

Usually systems consist of several interacting subsystems and such systems are referred to as *modular systems*. A modular system consists of a modular plant and a modular specification.

$$G \parallel K = (G_1 \parallel \cdots \parallel G_l) \parallel (K_1 \parallel \cdots \parallel K_m) \ . \tag{4.1}$$

In order to apply the synthesis algorithm described in Section 3.2.1, all the specifications are translated into plants. So, the synthesis problem consists of finding the least restrictive controllable and nonblocking supervisor for a set of plants,

$$G \parallel K^{\perp} = G_1 \parallel \cdots \parallel G_n. \tag{4.2}$$

To fight the state-space explosion problem, the compositional approach avoids building the complete monolithic state space, instead it tries to construct the monolithic model gradually. Before beginning the synchronisation process, each individual component is first simplified, and replaced by the equivalent component that the abstraction process yields. Synchronous composition is then computed step by step once individual abstractions are no more possible, iteratively abstracting again the intermediate results. Eventually, the procedure leads to a single automaton, which is an abstract description of the system. This automaton has less states and transitions than the original system. The final step is to use the final abstracted automaton to calculate a supervisor. Figure 4.1 illustrates the general compositional approach.

4.1.1 Local Events

The state space explosion problem is more noticeable when the components are loosely coupled. More to the point, some components have internal behaviors independent of others. While this independence can result in state space explosion, it can also be useful for developing the abstraction methodology in the compositional approach.

When abstracting an automaton G_i in the modular system (4.2), this automaton will typically contain some events that do not appear in the alphabet of any other components. These events are called *local events* and are denoted by the set Υ. Non-local or *shared* events are denoted by $\Omega = \Sigma \setminus \Upsilon$.

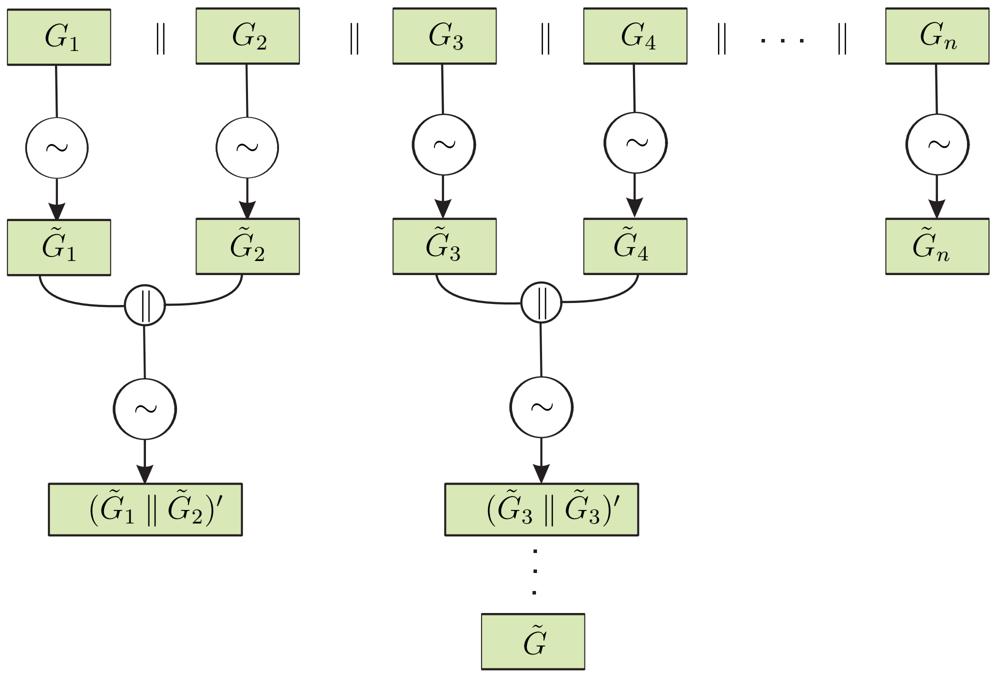
G_1 ∥ G_2 ∥ G_3 ∥ G_4 ∥ ⋯ ∥ G_n
∼
$\tilde{G}_1$
$\tilde{G}_2$
$\tilde{G}_3$
$\tilde{G}_4$
$\tilde{G}_n$
∥
$(\tilde{G}_1 \parallel \tilde{G}_2)'$
$(\tilde{G}_3 \parallel \tilde{G}_3)'$
$\tilde{G}$

Figure 4.1: General compositional approach. The modular system is described by $\{G_1, G_2, \cdots, G_n\}$ which is a set of plant automata and $\sim$ is a proper equivalence relation.

4.2 Equivalence Relations

Generally, the compositional approach attempts to replace individual components by an abstracted version. Such reasoning requires that the abstracted components are related to the original components. In this respect, a proper notion of equivalence needs to be identified. This can be done by defining the property that is required to be preserved. In this section some equivalence relations used for compositional synthesis are given. The intention of using a compositional approach here is to use the final result to calculate a supervisor to control a system. Thus, given a plant G and the supervisor S calculated in a monolithic way, in order to calculate $\tilde{S}$ (the supervisor calculated by the compositional approach) the equivalence relation could be defined as either maintaining an unchanged supervisor, $\mathcal{L}(S) = \mathcal{L}(\tilde{S})$, or having an unchanged closed loop behavior $\mathcal{L}(G \parallel S) = \mathcal{L}(G \parallel \tilde{S})$. Defining an equivalence relation considering supervisor equality was proposed and analysed in [15], and will be described in section 4.2.1.

However, supervisors are calculated to modify the closed loop behaviour of the system such that the specification is fulfilled. Consequently, in this thesis, maintaining an unchanged closed loop behavior is considered as the property of interest when calculating a supervisor. Furthermore, it gives us more freedom to abstract.

4.2.1 Supervision Equivalence

Typically in modeling a system by automata, each state of an automaton is labeled according to the state of the system. After synchronisation, the state label of global states is made up of combinations of their corresponding local states.

Earlier work on abstraction based compositional synthesis can be found in [15]. The final supervisor calculated in [15] is a set of state labels that defines the safe states in a symbolic way. The compositional approach proposed uses the information of the state labels to create a link between the symbolic supervisor and the original system. Thus, in this framework it is crucial to abstract each component so that the final symbolic supervisor has the same state labels as it would have had without any abstraction. Based on this property a notion of abstraction called *supervision equivalence* is introduced in [15]. When abstracting a component in the proposed compositional approach by collapsing states, the resulting merged state will get all the corresponding state labels. In the final step of the compositional approach, synthesis is applied on a single state that uses the same state labels as the original system.

The supervisor calculated in this framework is a least restrictive, controllable and nonblocking supervisor. However, the supervisor is monolithic. In addition, the equivalence requires additional state labels, making some desirable abstractions impossible, such as *bisimulation* described in Section 4.3. Thus, it is favorable to find an equivalence relation that is independent of state labeling.

4.2.2 Synthesis Abstraction

To maintain an unchanged closed loop behaviour after abstraction, *synthesis abstraction* is introduced in [Paper I]. Given a modular plant G, synthesis abstraction requires that the synthesis result for component G_1 and its abstraction $\tilde{G}_1$ are the same no matter what the behaviour of the remainder of the system is:

$$\begin{aligned} &\mathcal{L}(G_1 \parallel \cdots \parallel G_k \parallel \sup\mathcal{CN}(G_1 \parallel G_2 \parallel \cdots \parallel G_k)) \\ = &\mathcal{L}(G_1 \parallel \cdots \parallel G_k \parallel \sup\mathcal{CN}(\tilde{G}_1 \parallel G_2 \parallel \cdots \parallel G_k)) \ . \end{aligned} \tag{4.3}$$

Note that the supervisor calculated in this framework does not have the same language or number of states as the monolithic supervisor. However it produces the same closed loop language as the monolithic supervisor.

In this framework, the least restrictive *modular supervisor* consists of the final calculated supervisor and the specification K. However, the monolithic supervisor never needs to be calculated. It can be represented in its modular form, and synchronisation can be performed on-line, tracking the synchronous product states as the system evolves. In this way, synchronous product computation and state-space explosion can be avoided.

Example 8 *Consider automata G, $\tilde{G}$, and T in Figure 4.2. All events are controllable, and events α and β are local events since they only appear in G. States q_0 and q_1 in G can be merged resulting in $\tilde{G}$, which is a synthesis abstraction of G. Figure 4.2 shows $\tilde{S} = \sup\mathcal{CN}(\tilde{G} \parallel T)$ which has less states compared to the monolithic supervisor S which is also shown in Figure 4.2. However, both supervisor produce the same closed loop behaviour, that is, $\mathcal{L}(G \parallel T \parallel \tilde{S}) = \mathcal{L}(G \parallel T \parallel S)$.*

Several methods for abstracting automata such that synthesis abstraction is preserved are described in [Paper I]. However, to guarantee that the synthesis abstraction is preserved after applying these methods, the abstracted automata are required to be deterministic in all the abstraction steps.

Example 9 *Consider the modular system $\{G, T\}$ in Figure 4.3. Plant G in this system and the system in Example 8 differs in the transition between*

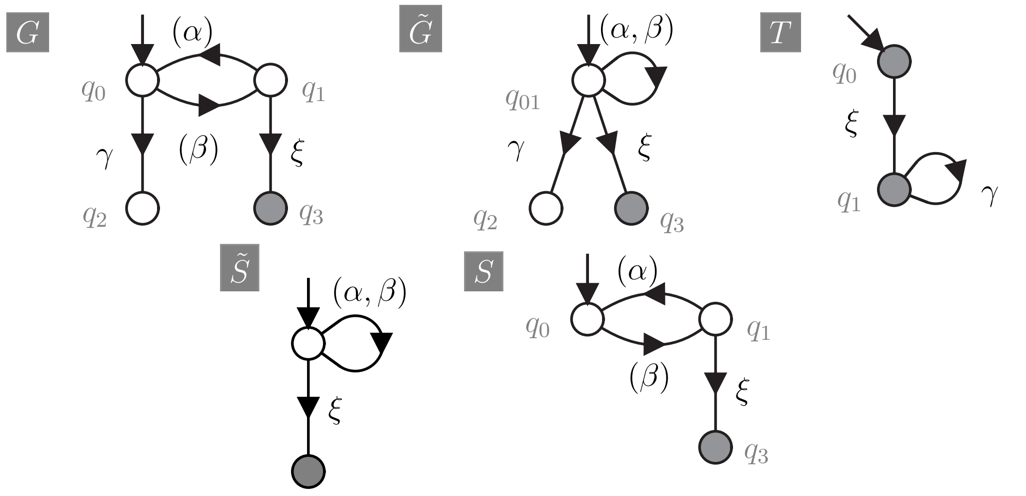

Figure 4.2: Automata of Example 8. Abstraction of G results in the deterministic automaton $\tilde{G}$.

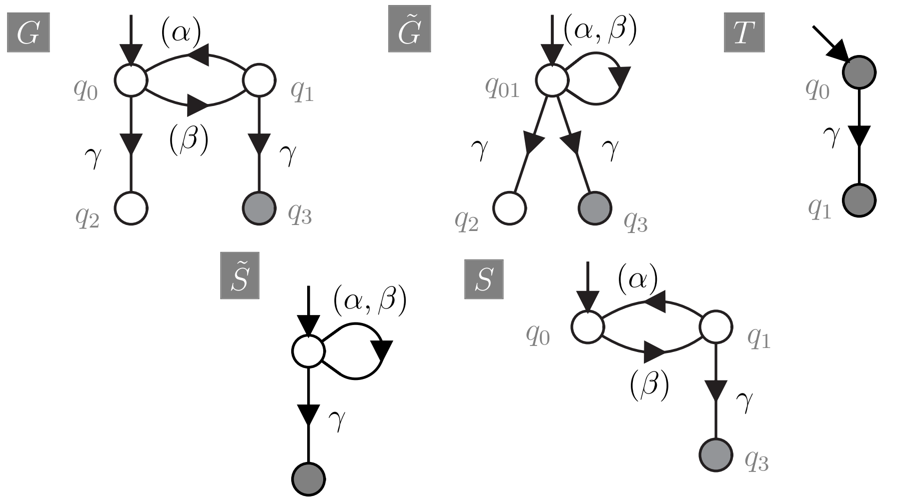

Figure 4.3: Automata of Example 9. Abstraction of G results in the nondeterministic automaton $\tilde{G}$, which is unsuitable as synthesis abstraction.

states q_1 and q_3. All events are controllable, and events α and β are local since they are only in G. The same abstraction method as in Example 8 results in merging states q_0 and q_1 in G. However, here merging these states results in a nondeterministic automaton $\tilde{G}$. Figure 4.3 shows $\tilde{S} = \sup\mathcal{CN}(\tilde{G} \parallel T)$. Since this supervisor enables event γ after both α and β, the closed-loop system is blocking, so $\tilde{S}$ is not a correct supervisor.

Even though the same abstraction method has been applied in both Example 8 and Example 9, synthesis abstraction has not been preserved in Example 9 due to nondeterminism.

Since synthesis abstraction requires deterministic automata after abstraction, the abstraction methods described in [Paper I] can not be applied when merging states results in nondeterminism. In order to be able to apply the abstraction methods regardless of nondeterminism, a new equivalence called *synthesis equivalence* was introduced in [Paper II].

4.2.3 Renaming and Synthesis Equivalence

Supervisory control theory is generalised for nondeterministic models in [18, 21,35] among others. In [18,21], even though the plant may be nondeterministic, the specification must be deterministic. This condition is relaxed in [35], where the plant and specification can be nondeterministic with the objective that the controlled system be bisimulation equivalent to the specification.

The nondeterminism considered in this work is the result of abstraction. To avoid nondeterminism after abstraction the idea of *distinguishing sensors* [5] is adapted and *renaming* is proposed in [Paper II]. Renaming introduces new events that are linked to the original nondeterministic transitions. After applying a renaming on one component, new events are introduced, so the remaining components need to be modified to use the new events.

Example 10 *Consider automata G and T in Figure 4.4, let $\mathcal{G}_0 = \{G, T\}$. As was seen in Example 9, merging q_0 and q_1 results in a nondeterministic automaton. To avoid nondeterminism, the renaming $\rho\colon \{\alpha, \beta, \gamma_1, \gamma_2\} \to \{\alpha, \beta, \gamma\}$ with $\rho(\alpha) = \alpha$, $\rho(\beta) = \beta$, and $\rho(\gamma_1) = \rho(\gamma_2) = \gamma$ can be applied on G, producing H. Since T also has γ, it needs to be modified into T'. Now abstracting H results in a deterministic automaton.*

When introducing renaming, some events are replaced by new events that do not appear in the original plant model. Then it is no longer clear how a supervisor synthesised from the renamed model can control the original plant. To make this possible, a *distinguisher* is introduced in [Paper II] that

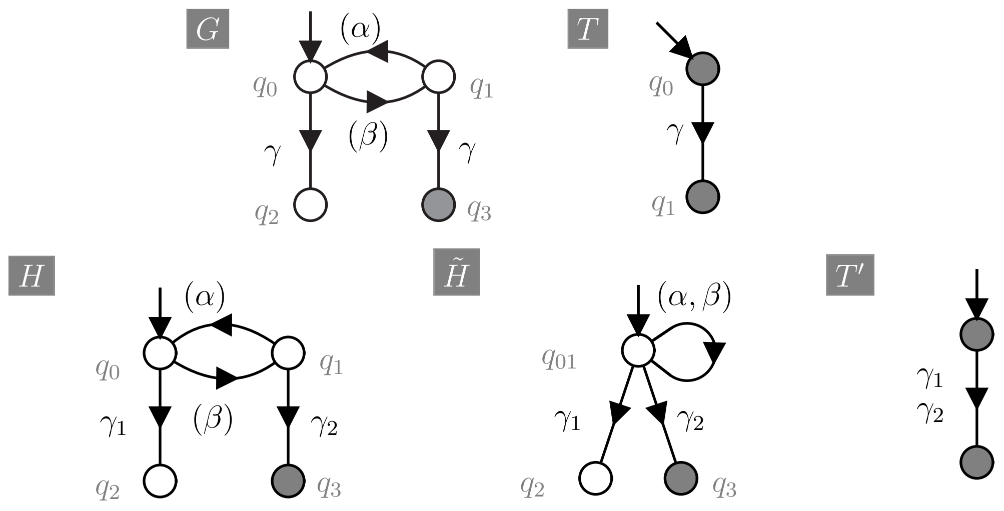

Figure 4.4: Automata of Example 10. Abstraction of G results in the non-deterministic automaton $\tilde{G}$ (see $\tilde{G}$ in Figure 4.3). Renaming event γ into γ_1 and γ_2 gives H, which leads to the deterministic abstraction $\tilde{H}$.

enables the final supervisor to choose the correct transitions. In Example 10, H is a distinguisher.

The original modular system (4.2) only consists of a set of uncontrolled plants, $\mathcal{G}$. In the new framework however, renaming and the corresponding distinguishers also need to be taken into account. To keep track of renaming, ρ, and the collected distinguishers, $\mathcal{S}$, a *synthesis triple*, written as $(\mathcal{G}, \mathcal{S}, \rho)$, is introduced in [Paper II]. Synthesis triples combine all the information collected at each abstraction step, and are the main data structure manipulated by the compositional synthesis algorithm.

Note that $\mathcal{S}$ not only contains distinguishers but also other components, referred to as *partial supervisors*. Section 4.3 will expand on the other components in $\mathcal{S}$.

To use the compositional algorithm for synthesis, a notion of abstraction for synthesis triples is needed. Every abstraction step must ensure that the synthesis result is the same as it would have been for the non-abstracted components. Based on this property, *synthesis equivalence* was introduced in [Paper II]. Two triples $(\mathcal{G}_1; \mathcal{S}_1; \rho_1)$ and $(\mathcal{G}_2; \mathcal{S}_2; \rho_2)$ are said to be synthesis equivalent, written $(\mathcal{G}_1; \mathcal{S}_1; \rho_1) \simeq_{\text{synth}} (\mathcal{G}_2; \mathcal{S}_2; \rho_2)$, if $\mathcal{L}(\rho_1(\sup\mathcal{CN}(\mathcal{G}_1) \parallel \mathcal{S}_1)) = \mathcal{L}(\rho_2(\sup\mathcal{CN}(\mathcal{G}_2) \parallel \mathcal{S}_2))$.

The compositional synthesis algorithm calculates a modular supervisor for a modular system $\mathcal{G} = \mathcal{G}_0$. Initially, no abstraction has been applied and no distinguisher or partial supervisor are collected yet. Thus, the initial distinguisher set is empty and the initial renaming is an identity renaming that maps each events to itself. At each step of the compositional approach, some automata are selected to compose, abstract and rename if necessary. After each step the renaming and the distinguisher are collected. These procedures change the initial triple iteratively such that synthesis equivalence is preserved. The algorithm terminates once there is a single automaton in the set of uncontrolled plants. This automaton represents the abstracted and renamed description of the original system. Finally, synthesis is applied on this automaton and the result is added to the supervisor set.

The final supervisor calculated in this framework is a set of supervisors. Here, since the synthesis triple contains the original uncontrolled plant, not only the closed loop behaviour is left unchanged, the synchronisation of the modular supervisor is equal to the monolithic supervisor.

Example 11 *Consider automata G and T in Figure 4.5, let $\mathcal{G}_0 = \{G, T\}$, and consider the initial synthesis triple $(\mathcal{G}_0; \emptyset; \mathrm{id})$. As suggested in Example 10, automaton G is replaced by H in Figure 4.5, using renaming ρ: $\{\alpha, \beta, \gamma_1, \gamma_2\} \to \{\alpha, \beta, \gamma\}$ with $\rho(\alpha) = \alpha$, $\rho(\beta) = \beta$, and $\rho(\gamma_1) = \rho(\gamma_2) = \gamma$. It can be shown that $(\mathcal{G}_0; \emptyset; \mathrm{id}) \simeq_{\mathrm{synth}} (\mathcal{G}_1; \mathcal{S}; \rho)$ where $\mathcal{G}_1 = \{H, T'\}$ and $\mathcal{S} = \{H\}$. Abstracting H results in the deterministic automaton $\tilde{H}$, shown in Figure 4.5. This changes the synthesis triple to $(\mathcal{G}_2; \mathcal{S}; \rho)$ where $\mathcal{G}_2 = \{\tilde{H}, T'\}$. The figure also shows $\tilde{S}_1 = \sup\mathcal{CN}(\mathcal{G}_2)$ which is a partial supervisor. The final modular supervisor contains the partial supervisor $\tilde{S}_1$ and the distinguisher H. Changing back the renamed events results in S, which is the least restrictive nonblocking and controllable supervisor.*

4.3 Abstraction Methods Preserving Synthesis Equivalence

Even though, it seems easy to define an equivalence relation that should be preserved, finding methodologies to simplify the systems in a way that the property of interest is preserved is not straightforward. The main challenge in the compositional synthesis approach is to find methods to abstract automata such that synthesis equivalence is preserved. Since the only step in the compositional approach that actually reduces the size of a system is the abstraction step, the efficiency of the compositional approach considerably depends on the abstraction methods.

Generally, abstraction methods for compositional verification such as those proposed in [14], are not applicable for compositional synthesis. In

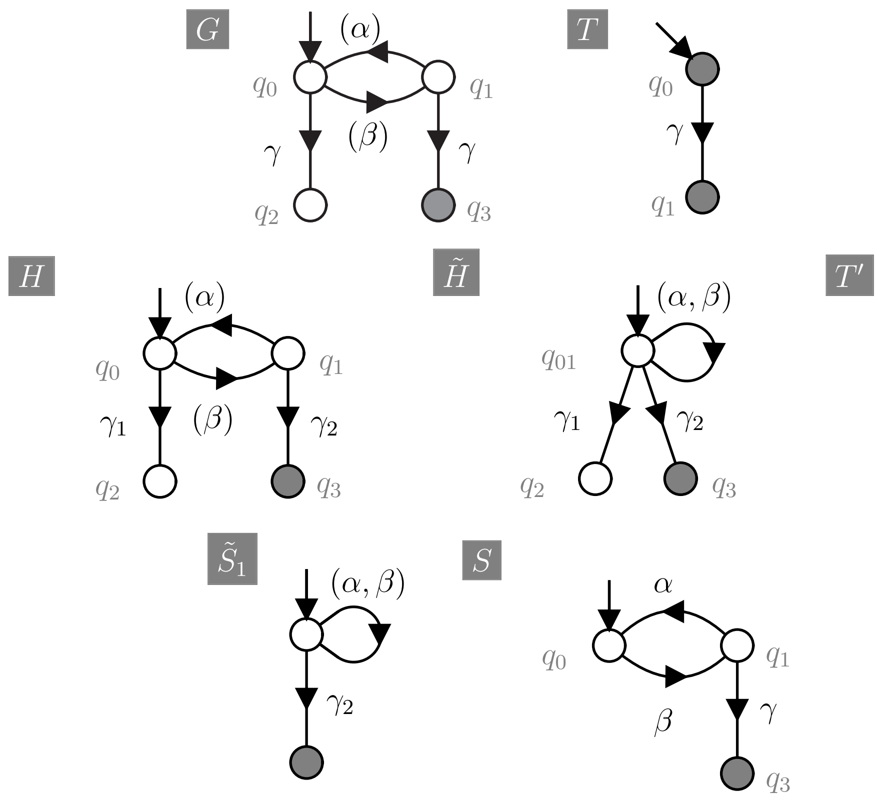

Figure 4.5: Automata of Example 11. Abstraction of G results in the nondeterministic automaton $\tilde{G}$ (see $\tilde{G}$ in Figure 4.3). To avoid nondeterminism, event γ in G is renamed into γ_1 and γ_2 producing H. Abstracting H leads to the deterministic $\tilde{H}$. Calculating a supervisor using $\tilde{H}$ and T' results in a least restrictive controllable and nonblocking supervisor $\tilde{S}_1 \parallel H$, which is equal to the monolithic supervisor.

compositional verification, the identity of local events can be disregarded. However, the events identity is needed in the compositional synthesis in order to enable the supervisor to make decisions. Moreover, less states can be merged in the compositional synthesis since merged states should not only have the same blocking property, they should also have the same synthesis property meaning either non of them are removed by synthesis or all of them are removed by synthesis.

This section discusses some possible methods to compute synthesis equivalent abstraction. Most of the abstraction methods presented in this section are based on *observation equivalence* [27]. While observation equivalence does not in general preserve synthesis equivalence, it can be strengthened to do so, as shown below.

Note that in [25, 30], observation equivalence is used in compositional synthesis. However, in these works the set of local events only consists of *unobservable* events and observable events are never considered as local. Yet, in this work *observable* events can also become local as soon as they only appear in one component. This makes it more difficult to find suitable abstractions, because the synthesised supervisor may synchronise on observable events, even if they are local.

4.3.1 Observation Equivalence

Observation equivalence or *weak bisimilarity* provides a well-known abstraction method [27]. Two states are considered observation equivalent if they are able to execute the same transitions when local events are not considered.

Definition 10 *Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton, and let $\Omega \subseteq \Sigma$. An equivalence relation $\approx \subseteq Q \times Q$ is called an* observation equivalence *on G with respect to Ω, if the following holds for all $x_1, x_2 \in Q$ such that $x_1 \approx x_2$: if $x_1 \xrightarrow{s_1} y_1$ for some $s_1 \in \Sigma^*$, then there exist $y_2 \in Q$ and $s_2 \in \Sigma^*$ such that $P_\Omega(s_1) = P_\Omega(s_2)$, $x_2 \xrightarrow{s_2} y_2$, and $y_1 \approx y_2$.*

Observation equivalence is known to preserve all temporal logic properties [27] and has been used in compositional verification [14]. However, it can be shown by a counterexample that observation equivalence in general does not preserve synthesis equivalence.

Example 12 *Consider the synthesis triple $\mathcal{G} = (\{G, T\}; \emptyset; \mathrm{id})$ where G and T are shown in Figure 4.6. Events α and β are controllable, while $!\upsilon$ is uncontrollable. Automata G and $\tilde{G}$ in Figure 4.6 are observation equivalent with respect to $\Upsilon = \{\alpha\}$. An attempt at abstracting $\mathcal{G} = (\{G, T\}; \emptyset; \mathrm{id})$ using $\tilde{G}$ gives $\tilde{\mathcal{G}} = (\{\tilde{G}_1, T_1\}; \{G_1\}; \mathrm{id})$, including the original automaton G_1*

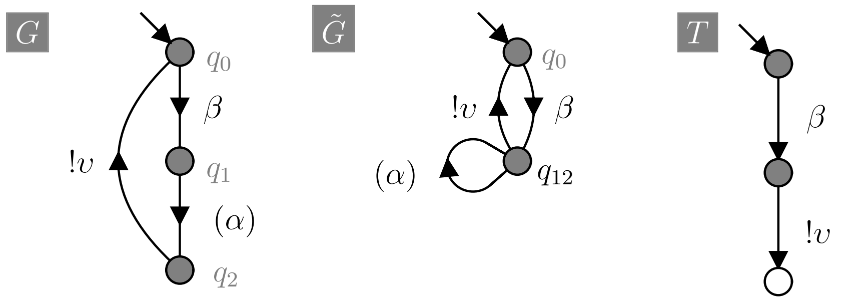

Figure 4.6: Automata of Example 12.

as a supervisor. However, $\beta \in \mathcal{L}(\sup\mathcal{CN}(\mathcal{G}))$ *and* $\beta \notin \mathcal{L}(\sup\mathcal{CN}(\tilde{\mathcal{G}}))$*, because with* G*, a supervisor could disable the local controllable event* α *to prevent entering state* q_2 *and thus the occurrence of the undesirable uncontrollable* $!\upsilon$*, but this is no longer possible in* $\tilde{G}$*.*

In the following, *bisimulation*, *uncontrollable observation equivalence* and *synthesis observation equivalence* are described as the methods preserving synthesis equivalence. These methods are obtained by restricting observation equivalence and were first proposed in [Paper I] to preserve synthesis abstraction. In [Paper III] it is proven that these methods also preserve synthesis equivalence.

Bisimulation

Example 12 shows that observation equivalence needs to be restricted if it is to be used in compositional synthesis. One simple way to restrict observation equivalence such that it implies synthesis abstraction is by not permitting any local events.

Observation equivalence without local events leads to *bisimulation equivalence* [27], one of the strongest known process equivalences. Two states are considered bisimilar if they have the same future behaviour. It has been shown in [Paper III] that merging bisimilar states results in synthesis equivalence.

Uncontrollable Observation Equivalence

While bisimulation ensures synthesis equivalence, not permitting any local events is highly restrictive, and it is desirable to relax the condition. In [Paper I] *uncontrollable observation equivalence* was introduced.

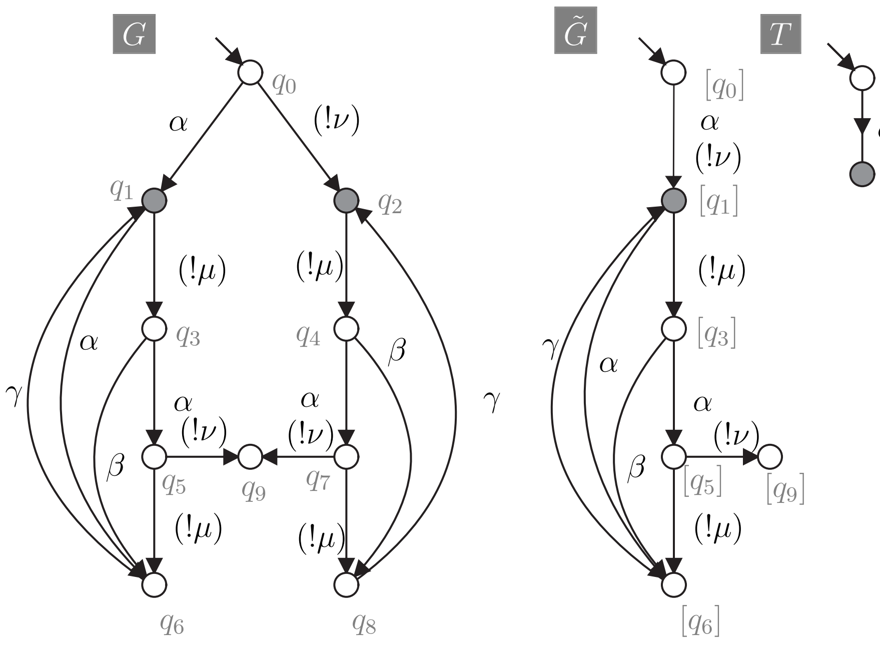

Figure 4.7: Automata of Example 13. $\tilde{G}$ is observation equivalent to G with only uncontrollable local events. Nevertheless it does not preserve synthesis equivalence.

In uncontrollable observation equivalence, like observation equivalence, the equivalent states need to have the same outgoing transitions while local events are disregarded. However, two more conditions need to be imposed to ensure that synthesis equivalence is preserved (Paper III, Definition 20). The first condition is,

restrict local events to be only uncontrollable.

The second condition is described by the use of the following example.

Example 13 *In Figure 4.7, the local events* $!\mu$ *and* $!\nu$ *are both uncontrollable, and* q_1 *and* q_2 *are observation equivalent. However,* $\alpha \in \mathcal{L}(\sup\mathcal{CN}(\tilde{G} \parallel T))$ *and* $\mathcal{L}(\sup\mathcal{CN}(G \parallel T)) = \emptyset$. *This is because the occurrence of* α *in the trace* $q_2 \xrightarrow{\mu\alpha\mu} q_8$ *results in the deadlock state* q_9 *after the uncontrollable* $!\nu$. *To avoid this situation* α *needs to be disabled in* q_4, *which makes* q_4 *a blocking state. Removing* q_4 *makes* q_2 *an uncontrollable state. However state* q_1 *is a safe state. Thus, though* q_1 *and* q_2 *are observation equivalent, merging them does not preserve synthesis equivalence.*

The problem in this example is that, the states q_1 and q_2 are considered equivalent even though q_1 is a safe state and q_2 is not. This difference is caused by considering $q_2 \xrightarrow{\mu\alpha\mu} q_8$, which is a sequence of unsafe states, as a matching sequence for $q_1 \xrightarrow{\alpha} q_6$ which is a safe transition. Thus, merging q_1 and q_2 does not result in synthesis equivalence. This problem can be avoided by requiring the following condition,

a trace matching a controllable transition must not contain any local events after the controllable event.

The second condition is added to guarantee that merging states have the same synthesis property.

Uncontrollable observation equivalence preserves synthesis equivalence as proved in [Paper III].

Synthesis Observation Equivalence

However, the conditions of uncontrollable observation equivalence can be relaxed, permitting controllable local events. In [Paper I] *synthesis observation equivalence* was introduced to preserve synthesis abstraction and in [Paper III] it is proven that it also preserves synthesis equivalence.

Synthesis observation equivalence is obtained by adding two extra requirements on observation equivalence (Paper III, Definition 21). The first requirement is,

a trace matching an uncontrollable transition must only contain uncontrollable events.

The second condition is described by the following example.

Example 14 *Automata G and $\tilde{G}$ in Figure 4.8 are observation equivalent with controllable local events α and β. In both G and $\tilde{G}$, the controllable event α must be disabled to prevent the undesired uncontrollable $!\nu$. By disabling α in G, state q_2 will be a blocking state and needs to be removed. Removing q_2 makes q_0 an uncontrollable state and it also needs to be removed. However disabling α does not make q_1 a blocking state. Thus, q_1 and q_2 can not be merged.*

The problem in Example 14 is that $q_1 \xrightarrow{\beta} q_4$, which is safe transition, is matched by $q_2 \xrightarrow{\alpha} q_5 \xrightarrow{\alpha} q_6$, which is not a safe sequence. In this example if either α was an uncontrollable event or q_5 was equivalent to q_2 or q_6, then $\tilde{G}$ would yield synthesis equivalence. Thus, the second condition that needs to be imposed is,

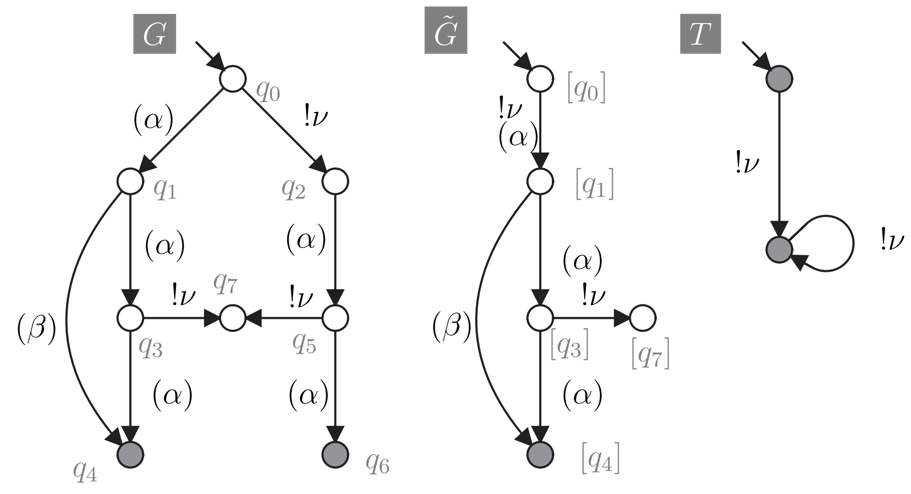

Figure 4.8: Automata of Example 14.

two states x and x' are equivalent if a local controllable transition $x \xrightarrow{\sigma} y$ have a matching sequence of local transitions $x' \xrightarrow{s} y'$ such that every state along this path, reached by a local controllable transition, is equivalent to x or y.

This condition ensures that all the equivalent states have the same synthesis property and removing one state, results in removing all of them.

Note that more abstraction is possible by synthesis observation equivalence than uncontrollable observation equivalence. However, uncontrollable observation equivalence has lower computational complexity and consequently abstraction can be done faster [Paper III].

4.3.2 Selfloop Removal

In the compositional verification, events used in only one component can immediately be removed from the model [14]. This is not always possible in compositional synthesis. Even if no other components use an event, the synthesised supervisor may still need to use it for control decisions that are not yet apparent. Therefore, events can only be removed if it is clear that no further supervisor decision depends on them.

An event can be removed from a synthesis triple if it causes no state change, which means that it appears only on selfloop transitions in the automaton. In this case, the event can be removed from all plant components. Plant components that disable a local selfloop need to be kept as partial supervisors in $\mathcal{S}$ to ensure that the final supervisor disables the event when needed. Selfloop removal preserves synthesis equivalence as proven in [Paper III].

4.3.3 Halfway Synthesis

Sometimes it is clear that certain states in a component must be removed in synthesis, no matter what the behaviour of the rest of the system is. Clearly, blocking states can never become non-blocking. Moreover, local uncontrollable transitions to blocking states must be removed, because it is clear that no other component nor the supervisor can disable a local uncontrollable transition.

Such states can be removed early on using *halfway synthesis* [15].

Definition 11 *Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ and $\Upsilon \subseteq \Sigma$. The* halfway synthesis result *for G with respect to Υ is*

$$\text{hsup}\mathcal{CN}_\Upsilon(G) = \langle \Sigma, Q \cup \{\bot\}, \rightarrow_{\text{hsup}}, Q^\circ \rangle \ , \tag{4.4}$$

where $\text{sup}\mathcal{CN}_\Upsilon(G) = \langle \Sigma, Q, \rightarrow_{\text{sup}}, Q^\circ \rangle$, $\bot \notin Q$, *and*

$$\rightarrow_{\text{hsup}} = \rightarrow_{\text{sup}} \cup \{\, (x, \sigma, \bot) \mid \sigma \in (\Sigma \cap \Sigma_u) \setminus \Upsilon,\ x \xrightarrow{\sigma},\ \text{and } x \xrightarrow{\sigma}_{\text{sup}} \text{ does not hold} \,\} \ . \tag{4.5}$$

Halfway synthesis is calculated like ordinary synthesis, but considering only events in the set Υ as uncontrollable. Potential controllability problems caused by other uncontrollable events are reflected in the result by including transitions to a blocking state, $\bot$.

Example 15 *Consider automaton G in Figure 4.9, which is part of a larger system. The uncontrollable events $!\mu$ and $!\upsilon$ are local. State q_3 is blocking, so q_2 is unsafe, because the uncontrollable $!\mu$-transition can not be disabled. Every nonblocking supervisor can and will disable the α-transition from state q_1 to q_2. State q_1 may still be safe, because some other component may disable the shared event $!\zeta$. The blocking state and the $!\zeta$-transition are retained in the halfway synthesis result* $\text{hsup}\mathcal{CN}_{\{!\upsilon,!\mu\}}(G)$, *see Figure 4.9, so a later synthesis is aware of the potential problem regarding $!\zeta$ in state q_1.*

Halfway synthesis preserves synthesis equivalence as proven in [Paper III].

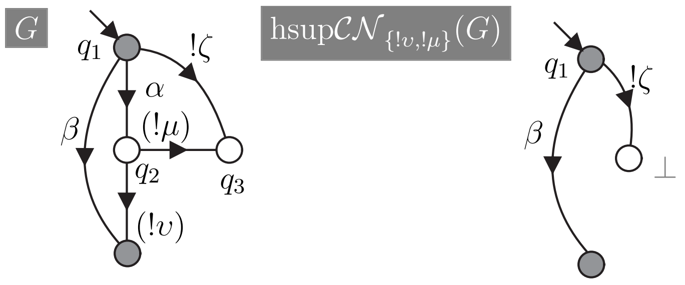

Figure 4.9: Automata of Example 15.

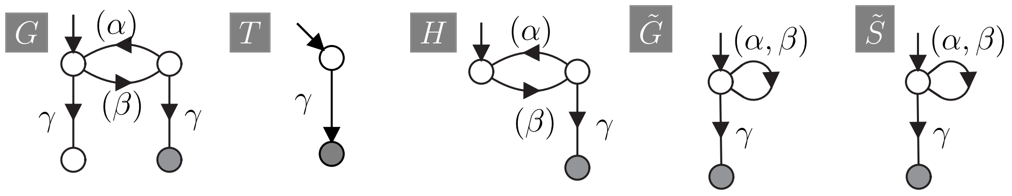

Figure 4.10: Automata of Example 16.

4.3.4 Partial Supervisors

Whenever the observation equivalence based abstraction are used, the original automata are kept as partial supervisor in $\mathcal{S}$. This is critical at the steps where observation equivalence based abstractions and halfway synthesis are applied simultaneously. This is illustrated in Example 16.

Example 16 *Consider the modular system $\mathcal{G} = \{G, T\}$ in Figure 4.10. All the events are controllable and events α and β are local events. First, halfway synthesis is applied on G, resulting in $H = \mathrm{hsup}\mathcal{CN}(G)$. Next, synthesis observation equivalence on H results in $\tilde{G}$. Considering $\tilde{S} = \mathrm{sup}\mathcal{CN}(\tilde{G} \parallel T)$ as the only supervisor results in a supervisor that allows events γ after both α and β. Allowing γ after α results in a blocking state. In order to enable the supervisor to make the correct decision H needs to be considered as a partial supervisor. By considering H to be a part of the supervisor, the supervisor only allows γ after event β.*

4.4 Experimental Results

The algorithms presented in this work has been implemented in DES software tool Supremica [31] and used to construct modular supervisors for a number of benchmark examples. The examples used here are the same examples that are used in [Paper III]. Here, the experimental results of [Paper III] are extended and also compared to the modular approach of Supremica.

4.4.1 Test Examples

The test cases include complex industrial models and case studies taken from various application areas such as manufacturing systems and automotive body electronics. All the test cases considered are either uncontrollable, blocking, or both. Some details about the test cases are given in the following.

agv Automated guided vehicle coordination based on the Petri net model in [28]. To make the example blocking in addition to uncontrollable, there is also a variant, **agvb**, with an additional zone added at the input station.

aip Automated manufacturing system of the Atelier Inter-établissement de Productique [7]. There are different version of aip. Here the early version aip0alps and a subsystem aip0sub1p0 are considered.

fencaiwon09 Model of a production cell in a metal-processing plant introduced in [12]. Here two variants are considered fencaiwon09b, which is blocking and fencaiwon09s, which is both blocking and uncontrollable.

fms2003 Large-scale flexible manufacturing system based on [36].

ftechnik Flexible production cell based on [24]; no controllable and non-blocking solution exists.

ipc Intertwined product cycles based on [23]. Two types of products are produced in a manufacturing system with two machines such that the products must move back and forth between the two machines in opposite directions.

tbed Model of a toy railroad system described in [22]. The original model is tbed-ctct. The blocking model tbed-nonderailb is created according to the original specifications and uncontrollable model tbed-uncont presents an other design.

verriegel Models of the central locking system of a BMW car. There are two variants, a three-door model **verriegel3**, and a four-door model **verriegel4**. These models are derived from [20].

For most of these models the monolithic approach fails to calculate a supervisor.

4.4.2 Heuristics

One crucial issue for compositional algorithms is to decide which automata to compose. The implementation presented here follows a two-step procedure introduced in [14]. In the first step a set of *candidates*, i.e., groups of automata that may be composed, is formed. In the second step the best candidate is identified. Both the set of candidates, and the “best” candidate are selected heuristically.

Some of the heuristics for the first step are:

- **minT** Candidates are all automata pairs containing the automaton with the fewest transitions.
- **mustL** For each event σ a candidate is formed by considering all automata with σ in the alphabet, so σ becomes a local event when composing the automata of the candidate.

The heuristics for the second step mostly rely on the fact that a high portion of local events increase the possibility of abstraction, and small composed automata increase the efficiency of the algorithm. The following heuristics can be used in the second step:

- **maxL** Choose the candidate with the highest portion of local events.
- **maxC** Choose the candidate with the highest portion of shared events.
- **minS** Choose the candidate for which the product of the number of states of the candidate automata is smallest.

4.4.3 Results

The algorithm described in [Paper III] has been used and the benchmarks were run on a standard desktop PC using a single core 2.66 GHz microprocessor.

In [Paper III] mustL/minS was selected as the best heuristic since persistently good results can be achieved for all the examples. However, here for different models different heuristics are selected. Different alternatives to select a heuristic is to optimise time, memory usage or the size of the largest supervisor. Since time and memory do not differ significantly, the heuristics are chosen based on the number of states of the largest supervisor component constructed.

The results of the experiments are shown in Table 4.1. For each model, the table shows the number of automata (Aut), the number of reachable states (Size) and whether the model is controllable or nonblocking. Next, the table shows the total runtime (Time), the total amount of memory used

Table 4.1: Compositional synthesis

					Time	Mem.	Supervisor		Heuristic	
Model	**Aut**	**Size**	**Nonb.**	**Cont.**	[s]	[MB]	**Num.**	**Largest**	**Step1**	**Step2**
agv	16	$2.6{\cdot}10^7$	true	false	0.25	382.7	6	2214	musL	maxC
agvb	17	$2.3{\cdot}10^7$	false	false	0.20	346.9	7	1680	musL	maxC
aip0	35	$3.0{\cdot}10^8$	false	true	0.73	370.2	3	18	musL	maxC
aip0sub1p0	46	24386	false	true	0.09	336.2	4	1	musL	maxL
fencaiwon09b	29	$8.9{\cdot}10^7$	false	true	0.39	389.9	9	1489	musL	maxC
fencaiwon09s	29	$2.9{\cdot}10^8$	false	false	0.39	110.0	11	2485	musL	minS
fms2003s	31	$1.4{\cdot}10^7$	false	true	27.82	111.1	6	156454	musL	minS
ftechnik	36	$1.2{\cdot}10^8$	false	false	0.01	109.2	0	0	musL	minS
ipc	12	20592	false	false	0.01	120.4	5	529	musL	minS
tbed-ctct	84	$3.1{\cdot}10^{12}$	false	true	74.50	144.3	0	0	musL	minS
tbed-noderailb	84	$3.1{\cdot}10^{12}$	false	true	4.6	282.3	11	784	minT	maxL
tbed-uncont	84	$3.6{\cdot}10^{12}$	true	false	5.5	381.8	12	4443	mustL	maxC
verriegel3b	52	$1.3{\cdot}10^9$	false	true	1.66	121.2	1	4	minT	maxC
verriegel4b	64	$6.2{\cdot}10^{10}$	false	true	4.5	131.8	1	4	minT	maxC

(Mem.), the number of supervisors (Num.) and the number of states of the largest supervisor component constructed (Largest).

All examples have been solved successfully with no more than two minutes runtime, and never using more than 400 MB of memory, even for models with more than 10^{12} reachable states. The largest supervisor component in all the cases except for **fms2003s** has no more than 5000 and the supervisor can be constructed efficiently.

It can be interesting to compare the compositional approach with the modular approach. The result comes from an implementation of the algorithm described in [1], available in Supremica. The result of this comparison can be found in Table 4.2. The compositional approach were run with different heuristics and for each model the heuristic that results in the smallest execution time is selected.

It can be seen from Table 4.2 that the modular approach for some cases is significantly faster than the compositional approach. This is because the modular approach only guarantees **local** nonblocking. To calculate a modular supervisor the connection between the specifications and the plants is exploited. The calculated supervisor is controllable but not necessarily globally nonblocking. An additional nonblocking check is needed to ensure nonblocking. It can be seen from Table 4.2 that the supervisor calculated by the modular approach in all the cases except for **agv** variations is globally blocking.

It can also be observed from Table 4.2 that for different variations of **tbed** and **aip0sub1p0** the modular approach fails to give a result within 5 minutes. The reason is that the modular approach for these models examines more states and transitions than necessary [13].

A close look at Table 4.1 and Table 4.2 reveals that when the heuristic is chosen based on the size of the supervisor, the best strategy to construct the candidate set is mustL, which ensures at least one local event in the composed automaton. However, when the heuristic is chosen based on time, the best heuristic to construct the candidate set is minT, which results in fewer transitions and consequently better time efficiency of the algorithm.

			Compositional		Modular	
Model	**Aut**	**Size**	**Time[s]**	**Heuristic**	**Time[s]**	**Nonb.**
agv	16	$2.6{\cdot}10^{7}$	0.25	mustL/maxC	0.08	true
agvb	17	$2.3{\cdot}10^{7}$	0.20	mustL/maxC	0.02	true
aip0alps	35	$3.0{\cdot}10^{8}$	0.39	minT/maxC	0.05	false
aip0sub1p0	46	24386	0.09	musL/maxL	-	-
fencaiwon09b	29	$8.9{\cdot}10^{7}$	0.14	minT/maxC	0.16	false
fencaiwon09s	29	$2.9{\cdot}10^{8}$	0.39	mustL/minS	0.06	false
fms2003s	31	$1.4{\cdot}10^{7}$	26.7	mustL/minS	0.08	false
ftechnik	36	$1.2{\cdot}10^{8}$	0.01	minT/maxC	0.13	false
ipc	12	20592	0.01	minT/maxC	0.06	false
tbed-ctct	84	$3.1{\cdot}10^{12}$	74.50	musL/minS	-	-
tbed-noderailb	84	$3.1{\cdot}10^{12}$	1.87	minT/maxC	-	-
tbed-uncont	84	$3.6{\cdot}10^{12}$	2.24	minT/maxC	-	-
verriegel3b	52	$1.3{\cdot}10^{9}$	1.63	minT/maxC	0.26	false
verriegel4b	64	$6.2{\cdot}10^{10}$	4.53	minT/maxC	0.16	false

Table 4.2: Compositional approach vs modular approach

Chapter 5

Summary of Included Papers

This chapter provides a brief summary of the papers that are included in the thesis.

Paper I

> Mohajerani, Sahar; Malik, Robi; Ware, Simon; Fabian, Martin: On the Use of Observation Equivalence in Synthesis Abstraction. *In Proceedings of the 3rd International Workshop on Dependable Control of Discrete Systems(DCDS'11)*, June 2011, Saarbrücken, Germany.

The work in this paper deals with strengthening observation equivalence to be applicable in a compositional synthesis. Three variations of observation equivalence are proposed that guarantee the construction of a correct modular supervisor. An example in the paper shows the suitability of these methods.

Paper II

> Mohajerani, Sahar; Malik, Robi; Fabian, Martin: Nondeterminism Avoidance in Compositional Synthesis of Discrete Event Systems. *In Proceedings of IEEE Conference on Automation Science and Engineering 2011(CASE'11), pp. 19-24*, August 2011, Trieste, Italy.

This paper generalised the compositional approach presented in [Paper I]. In this work *renaming* is introduced to make it possible to apply abstraction steps that were not possible in the previous paper due do nondeterminism.

Paper III

Mohajerani, Sahar; Malik, Robi; Fabian, Martin: Compositional Synthesis of Modular Supervisors Using Synthesis Equivalence. *Submitted to IEEE Transaction on Automatic Control, 2012.*

This paper combines and generalises results of the previous papers and proposes a general framework for calculating a least restrictive controllable and nonblocking supervisor. In addition, the paper adds formal proofs for the theorems, new abstraction rules and extensive experimental results.

Chapter 6

Concluding Remarks and Future Work

The state explosion problem is the main obstacle in calculating a supervisor. It has been proven in [17] that the general synthesis problem is NP-complete. However many systems are modular, which makes it possible to use the compositional approach and abstraction to reduce complexity of the system before synthesis.

The main contribution of this thesis is developing different kinds of abstractions that are guaranteed to preserve the final synthesis result, even when applied to individual components. These abstraction methods can considerably reduce the amount of states to examine, saving memory and time. Also in this work, renaming is introduced to avoid nondeterminism and its associated problems. The computed supervisor is modular in that it typically consists of several interacting components. The algorithm proposed in this work has been implemented in the DES software tool Supremica. Experimental results show that the method successfully computes modular supervisors for a set of large industrial models.

The compositional approach proposed here does not consider unobservable events in the models. It is likely that more abstractions are possible considering unobservable events in addition to local events.

The present framework does not support removing transitions besides local selfloops, since they can not be restored for control decisions. It would be interesting to generalise the present framework to support abstraction methods based on removing transitions.

Furthermore, finite-state machines augmented with bounded integer variables show good modelling potential, and it would seem useful to adapt the described compositional synthesis approach to work directly with this type of modelling formalism.

Bibliography

[1] K. Åkesson, H. Flordal, and M. Fabian. Exploiting modularity for synthesis and verification of supervisors. In *Proc. 15th IFAC World Congress on Automatic Control*, Barcelona, Spain, 2002.

[2] Knut Åkesson, Martin Fabian, Hugo Flordal, and Robi Malik. Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. In *Proc. 8th Int. Workshop on Discrete Event Systems, WODES '06*, pages 384–385, Ann Arbor, MI, USA, July 2006.

[3] A. Arnold. *Finite Transitions Systems: Semantics of Communicating Systems*. Prentice-Hall, 1994.

[4] A. Aziz, V. Singhal, G. M. Swamy, and R. K. Brayton. Minimizing interacting finite state machines: A compositional approach to language containment. In *Proc. Int. Conf. Computer Design*, 1994.

[5] G. Bouzon, M. H. de Queiroz, and J. E. R. Cury. Exploiting distinguishing sensors in supervisory control of DES. In *Proc. 7th Int. Conf. Control and Automation, ICCA '09*, pages 442–447, Christchurch, New Zealand, December 2009.

[6] B. A. Brandin, R. Malik, and P. Dietrich. Incremental system verification and synthesis of minimally restrictive behaviours. In *2000 American Control Conf.*, pages 4056–4061, Chicago, IL, USA, 2000.

[7] Bertil Brandin and François Charbonnier. The supervisory control of the automated manufacturing system of the AIP. In *Proc. Rensselaer's 4th Int. Conf. Computer Integrated Manufacturing and Automation Technology*, pages 319–324, Troy, NY, USA, 1994.

[8] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer, September 1999.

[9] C. G. Cassandras, S. Lafortune, and G. J. Olsder. Introduction to the modelling, control, and optimization of discrete event systems. In A. Isidori, editor, *Trends in Control*, pages 217–292. Springer, 1995.

[10] E. M. Clarke, D. E. Long, and K. L. McMillan. Compositional model checking. In *Proc. 5th IEEE Symp. Logic in Computer Science*, pages 353–362, 1989.

[11] J. E. R. Cury. Modular supervisory control of large scale discrete event systems. In R. Boel and G. Stremersch, editors, *Discrete Event Systems—Analysis and Control*, pages 103–110. Springer, 2000.

[12] Lei Feng, Kai Cai, and W. M. Wonham. A structural approach to the non-blocking supervisory control of discrete-event systems. *Int. J. Adv. Manuf. Technol.*, 41:1152–1168, 2009.

[13] Hugo Flordal. *Compositional Approaches in Supervisory Control.* PhD thesis, Chalmers University of Technology, Göteborg, Sweden, 2006.

[14] Hugo Flordal and Robi Malik. Compositional verification in supervisory control. *SIAM J. Control and Optimization*, 48(3):1914–1938, 2009.

[15] Hugo Flordal, Robi Malik, Martin Fabian, and Knut Åkesson. Compositional synthesis of maximally permissive supervisors using supervision equivalence. *Discrete Event Dyn. Syst.*, 17(4):475–504, 2007.

[16] A. Giua and F. DiCesare. Petri net structural analysis for supervisory control. *IEEE Trans. Robot. Autom.*, 10(2):185–195, April 1994.

[17] P. Gohari and W. M. Wonham. On the complexity of supervisory control design in the RW framework. *IEEE Trans. Syst., Man, Cybern.*, August 2000.

[18] Michael Heymann and Feng Lin. Discrete event control of nondeterministic systems, 1997.

[19] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[20] http://www4.in.tum.de/proj/korsys/.

[21] R. Kumar and M. A. Shayman. Centralized and decentralized supervisory control of nondeterministic systems under partial observation. *SIAM J. Control and Optimization*, 35(2):363–383, March 1997.

[22] R. J. Leduc. PLC implementation of a DES supervisor for a manufacturing testbed: An implementation perspective. Master's thesis, Dept. of Electrical Engineering, University of Toronto, ON, Canada, 1996.

[23] Feng Lin and W. Murray Wonham. Decentralized control and coordination of discrete-event systems with partial observation. *IEEE Trans. Autom. Control*, 35(12):1330–1337, December 1990.

[24] Annette Lötzbeyer and R. Mühlfeld. Task description of a flexible production cell with real time properties. Technical report, FZI, Karlsruhe, Germany, 1996.

[25] Petra Malik, Robi Malik, David Streader, and Steve Reeves. Modular synthesis of discrete controllers. In *Proc. 12th IEEE Int. Conf. Engineering of Complex Computer Systems, ICECCS '07*, pages 25–34, Auckland, New Zealand, 2007.

[26] Robi Malik and Hugo Flordal. Yet another approach to compositional synthesis of discrete event systems. In *Proc. 9th Int. Workshop on Discrete Event Systems, WODES '08*, pages 16–21, Göteborg, Sweden, May 2008.

[27] Robin Milner. *Communication and concurrency.* Series in Computer Science. Prentice-Hall, 1989.

[28] John O. Moody and Panos J. Antsaklis. *Supervisory Control of Discrete Event Systems Using Petri Nets.* Kluwer, 1998.

[29] Peter J. G. Ramadge and W. Murray Wonham. The control of discrete event systems. *Proc. IEEE*, 77(1):81–98, January 1989.

[30] Rong Su, Jan H. van Schuppen, and Jacobus E. Rooda. Model abstraction of nondeterministic finite-state automata in supervisor synthesis. *IEEE Trans. Autom. Control*, 55(11):2527–2541, November 2010.

[31] Supremica. `www.supremica.org`. The official website for the Supremica project.

[32] Simon Ware and Robi Malik. The use of language projection for compositional verification of discrete event systems. In *Proc. 9th Int. Workshop on Discrete Event Systems, WODES '08*, pages 322–327, Göteborg, Sweden, May 2008.

[33] Simon Ware and Robi Malik. Compositional nonblocking verification using annotated automata. In *Proc. 10th Int. Workshop on Discrete Event Systems, WODES '10*, pages 374–379, Berlin, Germany, 2010.

[34] W. M. Wonham and P. J. Ramadge. Modular supervisory control of discrete event systems. *Math. Control, Signals and Systems*, 1(1):13–30, January 1988.

[35] Changyan Zhou and Ratnesh Kumar. A small model theorem for bisimilarity control under partial observation. In *Proc. American Control Conf. 2005*, pages 3937–3942, Portland, OR, USA, August 2005.

[36] Meng Chu Zhou, Frank Dicesare, and Daryl L. Rudolph. Design and implementation of a Petri net based supervisor for a flexible manufacturing system. *Automatica*, 28:1199–1208, November 1992.

Part II
Publications