THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

On Compositional Approaches for Discrete Event Systems Verification and Synthesis

SAHAR MOHAJERANI

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2015

On Compositional Approaches for Discrete Event Systems
Verification and Synthesis
SAHAR MOHAJERANI
ISBN 978-91-7597-140-7

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 3821
ISSN 0346-718X

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
SE–412 96 Göteborg
Sweden

Telephone: +46 (031)772- 1000

Cover: Abstraction of a finite-state machine with 132 states to an equivalent finite-state machine with 12 states.

Typeset by the author using LaTeX.

Chalmers Reproservice
Göteborg, Sweden 2015

To my beloved Ali

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 modelled by *state machines*. Systems that are modelled in this way are referred to as *discrete event systems*. Usually, these systems are highly complex, and appear in settings that are safety critical, where small failures may result in huge financial and/or human losses. Having a control function is one way to guarantee system correctness.

The work presented in this thesis concerns *verification* and *synthesis* of such systems using the *supervisory control theory* proposed by *Ramadge* and *Wonham* [1]. Supervisory control theory 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 straightforward 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 as the number of states grows exponentially with the number of components, which makes it intractable to examine the global states of a system due to lack of memory and time.

One way to alleviate the state-space explosion problem is to use a compositional approach. A compositional approach exploits the modular structure of a system to reduce the size of the model. This thesis mainly focuses on developing abstraction methods for the compositional approach in a way that the final verification and synthesis results are 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 verify and compute memory efficient supervisors for several large industrial models.

Keywords: Finite-state machines, Extended finite-state machines, Verification, Synthesis, Abstraction, Compositional approach, Supervisory control theory.

Acknowledgements

In three days, I will be sending the thesis for print. This is the final piece of my thesis puzzle, and honestly the hardest part to write, because it has made me realise that my amazing journey as a PhD student is coming to an end. So I will take this last opportunity to try to thank the many wonderful people who have helped me on this journey. Ideally I would've preferred to mention everyone, but space does not allow me to.

My journey began when Martin Fabian believed in me when nobody else, even I myself, did not and for that I can eternally thankful. I cannot describe in words how happy I was when you gave me the news and I'll forever cherish that moment. During the past five years, whenever I felt stressed you calmed me down and I felt like I am talking to my friend instead of my supervisor. Your valuable supervision, encouragement, and support made this work possible. You are the most understanding and the greatest supervisor anybody could ask for and I am forever grateful to you!

After being a PhD student for six months, I had the great opportunity to meet Robi Malik, who then became my co-supervisor. I have visited Robi twice in New Zealand during my PhD studies. You have made the time I spent in New Zealand memorable and fruitful. You not only gave me great guidance and supervision but you also showed me amazing places in New Zealand and never let me feel homesick. I will always appreciate all the great discussions we had and I hope we can collaborate in future. Robi you are awesome and thanks heaps for everything!

I am also grateful to all my former and present colleagues and friends in the automation group for making the environment a really fun place to work in. Thanks to Bengt Lennartson, Knut Åkesson, Peter Falkman, Mona, Nina, Patrik and Sathya. I would specially like to thank Zhennan and Oskar for all the fun discussions, the cheerful memories and all their helps. I would also like to thank my Iranian friends at the Department of Signals and Systems; Roozbeh, Mitra, Azita and Maryam thank you for being great friends and cheering me up whenever I felt down. Big thanks go to my friends Sogol and Alexandra. You are always there for me and your friendship means a lot.

On the administrative side I would like to thank Lars Börjesson, Madeleine

Parsson and Christine Johansson for always being so helpful.

Finally, I would like to give my utmost gratitude to the sweetest people in my life, my family. To my mother, thank you for all your unconditional love. Sara, Maryam and Alireza, I love you guys so much and thanks for being fantastic sisters and brother. To my father in law, for being always supportive and understanding.

Now comes the most important person in my life, Ali. I could never have done this without you. Every amazing thing that has happened in my life is because of you. Your constant support and love has been there for me through the good and tough times. You are my best friend and my soulmate. Being with you is the *marked state* of my life and my life is never complete without you. I will always love you!

Sahar Mohajerani
Göteborg, February 2015

****

This work was supported by the Swedish Research Council, Vetenskapsrådet (VR).

List of publications

This thesis is based on the following three appended papers:

Paper 1

> **S. Mohajerani**, R. Malik, M. Fabian, "A Framework for Compositional Nonblocking Verification of Extended Finite-State Machines", *Invited paper to Journal of Discrete Event Dynamic Systems: Theory and Applications, special issue on WODES 2014*.

Paper 2

> **S. Mohajerani**, R. Malik, M. Fabian , "A Framework for Compositional Synthesis of Modular Nonblocking Supervisors", *IEEE Transaction on Automatic Control, vol. 59, no. 1, pp 150-162, 2014*.

Paper 3

> S. Mohajerani, R. Malik, M. Fabian, "Compositional Supervisor Synthesis with State Merging and Transition Removal", *Submitted to Automatica, 2014*.

Other publications

In addition to the appended papers, the following papers are also written by the author of the thesis:

(a) **S. Mohajerani**, J. Sjöberg "On Initialization of Iterative Algorithms for Nonlinear ARX Models", *In proceeding of the 8th IFAC Symposium on Nonlinear Control Systems*, September 2010, pp. 362-367.

(b) **S. Mohajerani**, R. Malik, S. Ware, M. Fabian, "Three variations of observation equivalence preserving synthesis abstraction", University of Waikato, Department of Computer Science, Hamilton, New Zealand, *Technical Report*, January, 2011.

(c) **S. Mohajerani**, R. Malik, S. Ware, M. Fabian, "Compositional Synthesis of Discrete Event Systems Using Synthesis Abstraction", *In Proceeding of the 23th Chinese Control and Decision Conference*, May 2011, pp. 1549-1554.

(d) **S. Mohajerani**, R. Malik, S. Ware, M. Fabian, "On the use of observation equivalence in synthesis abstraction", *In Proceeding of the 3rd International Workshop on Dependable Control of Discrete Systems (DCDS)*, June 2011, pp. 84-89.

(e) **S. Mohajerani**, R. Malik, M. Fabian, "Nondeterminism avoidance in compositional synthesis of discrete event systems", *In Proceeding of the 7th IEEE International Conference on Automation Science and Engineering*, August 2011, pp. 19-24.

(f) **S. Mohajerani**, R. Malik, S. Ware, M. Fabian, "Synthesis observation equivalence and weak synthesis observation equivalence", University of Waikato, Department of Computer Science, Hamilton, New Zealand, *Technical Report*, July, 2012.

(g) **S. Mohajerani**, R. Malik, S. Ware, M. Fabian, "Five abstraction rules to remove transitions while preserving compositional synthesis results", University of Waikato, Department of Computer Science, Hamilton, New Zealand, *Technical Report*, July, 2012.

(h) **S. Mohajerani**, R. Malik, M. Fabian, "Transition removal for compositional supervisor synthesis", *In Proceeding of the 8th IEEE International Conference on Automation Science and Engineering*, August 2012, pp. 694-699.

(i) **S. Mohajerani**, R. Malik, M. Fabian, "An algorithm for weak synthesis observation equivalence for compositional supervisor synthesis", *In Proceeding of the 11th International Workshop on Discrete Event Systems*, October 2012, pp. 239-244.

(j) **S. Mohajerani**, R. Malik, S. Ware, M. Fabian, "Partial unfolding for compositional nonblocking verification of extended finite-state machines", University of Waikato, Department of Computer Science, Hamilton, New Zealand, *Technical Report*, January, 2013.

(k) **S. Mohajerani**, R. Malik, M. Fabian, "Compositional nonblocking verification for extended finite-state automata using partial unfolding, *In Proceeding of the 9th IEEE International Conference on Automation Science and Engineering (CASE)*, August 2013, pp. 930-935.

(l) S. Ware, R. Malik, **S. Mohajerani**, M. Fabian, “Certainly Unsupervisable States”, *In Proceeding 2nd International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS 2013)*, October 2013, pp. 3-18.

(m) **S. Mohajerani**, R. Malik, M. Fabian, “An algorithm for compositional non-blocking verification of extended finite-state machines”, *In Proceeding of the 12th International Workshop on Discrete Event Systems (WODES’14)*, October 2014, pp. 376-382.

Contents

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 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 requirement 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*. A coffee machine, a printer and industrial robots are some examples of systems.

When dealing with different systems, many questions about the properties of the systems arise. 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 system behaviour.

Modelling is done from different perspectives. In some cases, 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. In other cases, a system can be viewed as *event-driven*, for example, when a coffee machine goes out of coffee beans, the *event*, it goes from a working *state* to an idle state. The behaviour of such a system can then be described by sequences of events, where the occurrence of an event causes the system to go from one state to another. Such system models are referred to as *discrete event systems* and are the main focus of this thesis. In order to model a discrete event system, intuitive formalisms such as *finite-state machines* and *extended finite-state machines* can be used.

1.1 Problem Statement

Imagine a coffee machine that fills your glass with tea even though you asked for coffee. In this case you may just accept the tea, get back to work and be in a bad mood all day. However, many applications of discrete event systems take place in settings that are safety critical and small failures may result in huge financial and/or human losses. Moreover, as discrete event systems are usually complex, their development is error-prone. Thus we need to verify that a system is error-free or if there are errors, remove them before using the system.

In this thesis *formal verification* is used to approve or disprove the correctness of a system. In formal verification, the first step is to identify a desired property. Then, a model of the system is built and finally it is shown mathematically whether the property of interest is fulfilled or not. Thus, the final result after verification is either “yes” or “no”.

In the case that the verification result is not satisfactory, the next step is to design a control function to guarantee system correctness.

In 1989, *Ramadge* and *Wonham* [1] 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, the *specification*, the supervisory control theory proposes methods to design a *supervisor* in such a way that the closed-loop system of plant and supervisor always acts according to the specification.

For simple systems consisting of a small number of states, verification or supervisor calculation can be done straightforwardly. However, this is not viable for complex systems consisting of several interacting subsystems. Such systems are referred to as *modular systems*. Using the straightforward approach to verify or calculate a supervisor for these 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 verify and calculate a supervisor is to go through all states and verify the property of interest for each particular state and remove undesirable states. However, 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 typically occurs 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 a *compositional approach*. To avoid the state-space explosion problem, a compositional approach tries to build a single representation of a system in an iterative way. The general approach is as follows. First the subsystems are simplified in such a way that the property of interest is

preserved. When no further simplification (also called *abstraction*) is possible, 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 verification or synthesis.

1.2 Main Contributions

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

- What considerations need to be taken into account when the compositional approach is used for different modelling formalism?
- What considerations need to be taken into account when the compositional approach is used for verification and for synthesis?
- Is it possible to find methods in order to efficiently simplify subsystems?
- In the case that the compositional approach is used for supervisor calculation, does the supervisor have a modular structure and is it memory efficient?

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

- The compositional approach is well-developed for verification of systems modelled as finite-state machines [2]. This framework is extended to consider systems that are modelled as extended finite-state machines. It is shown how the abstraction methods defined for finite-state machines can be applied on extended finite-state machines (Paper 1).
- When using the compositional approach for supervisor calculation or verification, the property of interest is different and thus needs to be defined first before using the compositional approach. *Conflict equivalence* [2] is used as the property to be preserved when the task is to verify whether the system is able to finish some sub-tasks and it is used in Paper 1. When it comes to supervisor calculation, the closed-loop behaviour is the property to be preserved after simplification. For this purpose *synthesis equivalence* is introduced in this thesis (Paper 2 and 3).
- The main focus of this thesis is to develop abstraction methods in the compositional approach such that the property of interest is preserved. It is shown how any abstraction method defined for finite-state machines can

be applied on extended-finite state machines (Paper 1). Different abstraction methods for compositional synthesis are presented, which are mostly based on a well known abstraction method called *observation equivalence*. It is shown how observation equivalence can be strengthened to be applicable in the compositional synthesis framework (Paper 2 and 3).

- The algorithms proposed in this work have been implemented in the discrete event system software tool Supremica and have been applied to compute supervisors for several benchmark examples. The experimental results show that the method efficiently computes modular supervisor for a set of very large industrial models (Paper 2 and 3). The supervisor can also be represented in a compact form and can be stored efficiently (Paper 3).

1.3 Outline

The first two chapters, Chapter 2 and Chapter 3, give the preliminaries and background of the supervisory control theory. In Chapter 4, the compositional verification and synthesis proposed in this work is described. The summary of appended paper is provided in Chapter 5. Finally some concluding remarks and future work are given in chapters 6 and 7.

Chapter 2

Preliminaries

The behaviour of many technical devices and systems in common use can be described by sequences of events, for example a robot arm picking up a workpiece. This includes automated manufacturing systems, traffic control systems, etc. The behaviour of these systems can be modelled 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 the occurrence of an event.

2.1 Modelling Formalism

A prerequisite to formally analyse discrete event systems is developing suitable models that can accurately represent the activities of the system. Different modelling formalisms have been used in the literature, for instance, state machines [3], Petri nets [4], process algebra [5] and formal languages [1, 6].

In this thesis, *finite-state machines* are used to represent the behaviour of discrete event systems as these are intuitive and have structure that allows useful manipulations. For example, abstraction may cause nondeterministic behaviour, and state machines describe nondeterministic behaviour straightforwardly.

2.1.1 Finite-State Machines

Finite-state machines (FSM), referred to as *finite-state automata* in Paper 2 and 3, are devices that represent the behaviour of discrete event systems. An FSM can be considered as a directed graph. A state represents the current status of a system under which certain conditions hold, such as the position of a robot arm. The state set of a system contains all possible situations that the system may encounter. Events represent incidents that cause transitions from one state to another. For a discrete event system, a finite alphabet Σ is defined, the elements

of which are all the possible events in the system. A sequence of events forms a *string*, and Σ^* is the set of all finite strings of events from Σ. Transitions of FSMs are written as $x \xrightarrow{\sigma} y$, where x is the *source state*, and y is the *target state* reached after the occurrence of the event σ. Two more ingredients are necessary to define an FSM: *initial states* and *marked states*. The system starts in one of the initial states. Marked states are desired states with a special meaning attached to them like completion of a task. In the figures, initial states are identified by an arrow pointing into them, and the marked states are shaded grey.

Usually, systems have a unique initial state, and each occurrence of an event in a given state x causes a transition to only one state y, and all the transitions are labelled by events from the alphabet of the system. Under these conditions, the system is said to be *deterministic*, as the state of the system can be uniquely determined from the sequence of events that have occurred. However the main focus of this thesis is abstraction, which may cause nondeterminism. Moreover, events that represent *internal* behaviour of a component in a system are removed from the alphabet and the transitions labelled by those events are labelled by the *silent event* τ. This event is not part of the alphabet of the system, but its use is explicitly mentioned by the notation $\Sigma_\tau = \Sigma \cup \{\tau\}$. The act of transforming an event into the silent event is referred to as *hiding* [7] and introduces nondeterminism. The formal definition of hiding can be found in Def. 3 of Paper 3.

Now we can state a formal definition for a finite-state machine.

Definition 1 *A finite-state machine (FSM) is a tuple* $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$, *where*

- Σ *is the* alphabet, *a finite set of events,*
- Q *is the finite set of* states,
- $\rightarrow \subseteq Q \times \Sigma_\tau \times Q$ *is the* state transition relation,
- $Q^\circ \subseteq Q$ *is the set of* initial states,
- $Q^\omega \subseteq Q$ *is the set of* marked states.

An FSM G is *deterministic* if $|Q^\circ| \leq 1$, meaning it has at most one initial state, $x \xrightarrow{\sigma} y_1$ and $x \xrightarrow{\sigma} y_2$ always implies $y_1 = y_2$, meaning that occurrence of an event in a source state leads the system to a unique target state, and $x \xrightarrow{\sigma} y$ implies $\sigma \neq \tau$, meaning transitions are only labelled by events from the alphabet. In this thesis, $Q^\circ \xrightarrow{\sigma} y$ means there exists $x^\circ \in Q^\circ$ such that $x^\circ \xrightarrow{\sigma} y$ and $x \rightarrow y$ means there exists $\sigma \in \Sigma$ such that $x \xrightarrow{\sigma} y$. Moreover, $x \xrightarrow{s}$ means that $x \xrightarrow{s} y$ for some $y \in Q$.

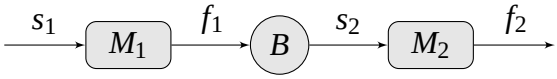


Figure 2.1: Manufacturing system overview.

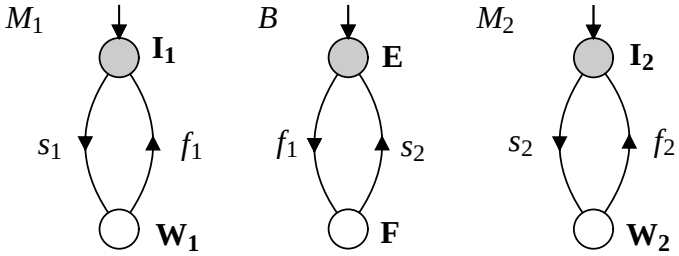


Figure 2.2: FSM models of manufacturing system.

Normally a system is modelled as a set of FSMs where each FSM represents the behaviour of an individual component of the system. This makes the modelling task easier as it is more intuitive to describe each component behaviour rather than the entire system at once.

Example 1 *Consider the simple manufacturing system shown in Fig. 2.1. The system consists of two machines M_1 and M_2, which are linked by a buffer B that can store one workpiece. The first machine M_1 takes workpieces from outside the system (event s_1), processes them, and puts them into B (event f_1). Machine M_2 takes workpieces from B (event s_2), processes them and outputs them from the system (event f_2). Fig. 2.2 shows FSMs modelling the system.*

2.1.2 Extended Finite-State Machines

Finite-state machines describe the behaviour of a system using states and events. For systems with data dependency, it is natural to extend finite-state machines with variables and updates. This results in *extended finite-state machines (EFSM)*, also referred to as *extended finite-state automata* [8].

EFSMs are similar to conventional finite-state machines, but the transitions are not only labelled by events, but also by *updates* [8–12]. Updates are predicates and can be evaluated to **T** or **F**. They are constructed from variables, integer constants, the Boolean literals *true* and *false*, and the usual arithmetic and logic connectives. Similar to FSMs, a system changes its state on the occurrence of an event, but the transition in an EFSM is enabled only if the corresponding update evaluates to **T**. The transitions of an EFSM are represented as $x \xrightarrow{\sigma:p} y$, where x is the *source location*, and y is the *target location* after the occurrence of the event σ, and p represents the update associated to the transition. Once a transition

occurs, the system moves from the source location to the target location, and the variables in the update of the transition may change their value, while the rest of the variables remain unchanged. Thus, the states of an EFSM are combinations of locations of the EFSM and the variable values.

As mentioned, updates are constructed from variables. Each variable has a discrete domain, $\mathrm{dom}(v)$, that represents the possible values of the variable. For example, if a buffer with capacity 3 is represented as a variable b in the system then the domain of b is $\{0,1,2,3\}$, where each value represents the number of workpieces in the buffer. A variable also has an initial value, $v^\circ \in \mathrm{dom}(v)$. For example, if the buffer is initially empty then $b^\circ = 0$. As mentioned, when a transition occurs, the values of the variables in the corresponding update may change while the variables not in the update remain unchanged. To distinguish the changing variables, a second set of variables called *next-state variables*, denoted by V', is used, which have the same domain as the variables in V. For example, again assume we have a buffer with capacity 3 represented by the variable b, a transition with update $b' = b+1$ adds a workpiece to the buffer, if the number of workpieces in the buffer is currently less than 3. Otherwise, if $b = 3$, then the buffer is full and the transition is disabled since no more workpieces can be added. In the model this is detected since the value of b' would become equal to 4, which is outside of the domain of b. An update like $b = 3$ simply checks if the number of workpieces in the buffer is 3 and enables the transition only if this is true. In this case the value of b in the target location remains unchanged as no workpiece is added to the buffer. Differently, an update like $b' = 3$ always enables its transition, and the value of b in the target location is forced to be 3.

In the figures of this thesis, for simplicity updates only constructed from *true* are not shown on transitions of EFSMs.

Definition 2 *An* extended finite-state machine (EFSM) *is a tuple* $E = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$, *where* Σ *is a set of events,* Q *is a finite set of* locations, $\rightarrow \subseteq Q \times \Sigma \times \Pi_V \times Q$ *is the* conditional transition relation *and* $p \in \Pi_V$, *where* Π_V *contains all possible updates over the variable set* V, $Q^\circ \subseteq Q$ *is the set of* initial locations, *and* $Q^\omega \subseteq Q$ *is the set of* marked locations.

In this thesis, the term *state machine* is used to refer to both finite-state machine and extended finite-state machine.

EFSMs usually simplify the modelling task. However, to analyse EFSM systems the straightforward way would be to convert the EFSM to an FSM by evaluating all the updates to find the variable values in each location of the EFSM. The states of the resultant FSM are then combinations of the locations of the EFSM and the variable values. This is referred to as *unfolding* variables. The complete process of transforming an EFSM to FSM is called *flattening*, and the

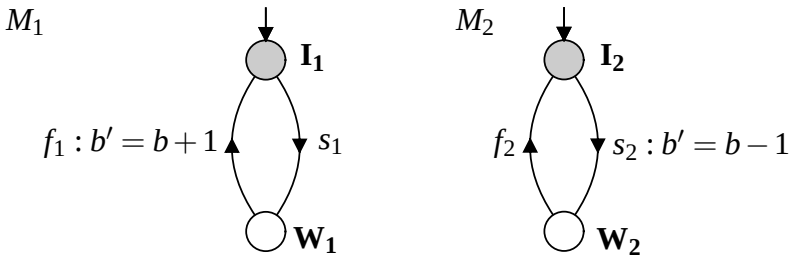


Figure 2.3: EFSM models of the manufacturing system.

resultant FSM is referred to as the *flattened FSM*. At the beginning of the flattening process, the value of each variable is equal to its initial value. Thus, the initial locations x° are combined with the initial values of the variables, denoted as $\hat{v}^\circ$, to create the initial states of the form $(x^\circ, \hat{v}^\circ)$. Next, based on the updates of the transitions going out of the initial locations, the values of some of the variables change at the next locations, and so on. Thus, for each transition $x \xrightarrow{\sigma:p} y$ in the EFSM model there exist transitions $(x,\hat{v}) \xrightarrow{\sigma} (y,\hat{w})$ in the flattened FSM model whenever p evaluates to **T** for the values $\hat{v}$ and $\hat{w}$. The formal definition of flattening can be found in Def. 10 of Paper 1.

Example 2 *Consider again the manufacturing system shown in Fig. 2.1. The EFSM model of the system consists of M_1 and M_2 as shown in Fig. 2.3. It uses a variable b with domain $\{0,1\}$ to represent the number of workpieces in the buffer. The update $b' = b+1$ represents an addition of a workpiece to the buffer when the event f_1 is executed. As the domain of b is $\{0,1\}$, event f_1 can only be executed if the current value of b is zero, or in other words when there is no workpiece in the buffer. If the buffer is full, then $b = 1$ and $b' = 1+1 = 2$, which cannot happen as 2 is not in the domain of b. Thus, if the buffer is full, M_1 cannot add another workpiece to the buffer.*

2.2 Interaction

As mentioned before, discrete event systems are usually *modular*, in that they are modelled as a set of interacting subsystems. The reason is that typically systems are complex and modelling them by only one state machine is impractical. However, it is possible to combine the state machine components of a system into a single state machine. This process is referred to as *synchronous composition* [13].

When a system is modelled as a set of interacting state machines, a transition in the synchronous composition occurs only if it is possible in all the components *sharing* the event labelling the transition, otherwise the transition is disabled. Af-

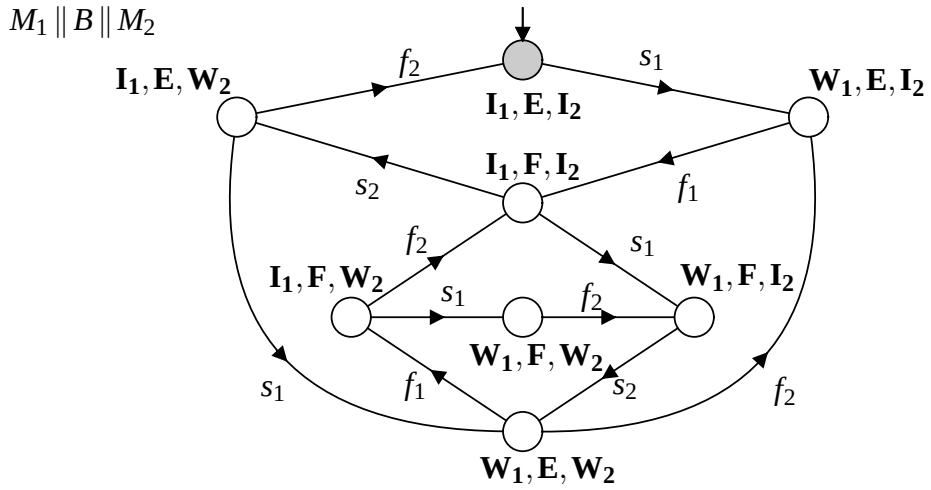


Figure 2.4: Synchronous composition of FSM model of the manufacturing system.

ter an occurrence of a shared event, the states (locations) of all the state machines with that event in their alphabet are updated concurrently. If an event only appears in one component then this event is called a *local event* and it is always executed independently. Transitions of EFSMs are in addition to events also labelled by updates. Thus, updates need to be considered during synchronisation. The updates of the shared events are combined by *conjunction* while the updates of local events remain unchanged. Using these principles it is possible to build a single state machine that represents the behaviour of a set of interacting state machines. The formal definition of synchronisation of FSMs is given in Def. 2 in Paper 2 and 3, and of EFSMs in Def. 9 of Paper 1.

After the synchronisation of G_1 and G_2, in the worst case the number of the states (locations) of the synchronisation result of G_1 and G_2 is $|Q_1| \times |Q_2|$. Thus, the state-space of systems consisting of many interacting components may easily become unmanageable. This problem is commonly referred to as the *state-space explosion* problem.

Example 3 *Fig. 2.2 shows the model of the small manufacturing system of Example 1. Initially the machines and the buffer are in their respective I and E states. Thus, the initial state of the synchronous composition is* (I_1, E, I_2)*. In this state, only the local event* s_1 *is possible. Note that* f_2 *and* s_2 *are enabled in states E and* I_2 *respectively. However, they are not enabled in* (I_1, E, I_2) *as they are restricted by the other components. After the occurrence of event* s_1*, the first machine moves to the* W_1 *state and the buffer and the second machine remain in their respective E and* I_2 *states. The entire synchronous composition is shown in Fig. 2.4. Fig. 2.5 shows the synchronous composition of the EFSM model. The updates in the synchronisation result are the same as in* M_1 *and* M_2 *because*

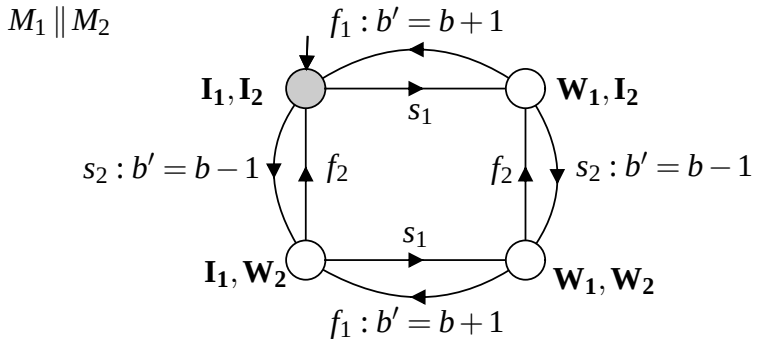


Figure 2.5: Synchronous composition of the EFSM model of the manufacturing system.

these two EFSMs do not share any events. Flattening the synchronised EFSM $M_1 \,\|\, M_2$ results in an FSM isomorphic to FSM $M_1 \,\|\, B \,\|\, M_2$ shown in Fig. 2.4.

2.3 Event-Based Marking

Marking is used to represent states of a system that have a special meaning attached to them, like completion of a task. The standard way to represent the marking in a system is by labelling some states (locations) as *marked* [1, 6]. This thesis is mainly focused on equivalence relations to abstract state machines. For this purpose, in Paper 2 and 3 marking of states of FSMs is transformed to an event-based representation, otherwise marking needs special consideration in most of the definitions in these papers. For this, a *termination event* ω is introduced. This event, similarly to the silent event, is not included in the alphabet of the system, but its use is explicitly mentioned by the notation $\Sigma_\omega = \Sigma \cup \{\omega\}$.

Consider an FSM with Q^ω as the set of marked states and the alphabet Σ. To transform this to event-based marking, the first step is to add to the set of states a termination state $\perp \in Q \setminus Q^\omega$. This state has no outgoing transitions and is not originally in Q. After adding the termination state, the transition relation is extended to $\rightarrow \subseteq Q \times \Sigma_\omega \times Q$ by adding transitions

$$q^\omega \xrightarrow{\omega} \perp \text{ for each } q^\omega \in Q^\omega \tag{2.1}$$

In synchronous composition, ω is considered as an ordinary event. Thus, for a composed state (x_1, x_2), it holds that $(x_1, x_2) \xrightarrow{\omega} \perp$ only if $x_1 \xrightarrow{\omega} \perp$ and $x_2 \xrightarrow{\omega} \perp$. For simplicity in the figures, marked states are shown instead of the ω and $\perp$.

2.4 Equivalence Relations

One way to alleviate the state-space explosion problem is to *abstract* the components of the system by merging some states or removing some transitions. To merge states it is important that the states are equivalent based on some criteria that preserves the property of interest. An *equivalence relation* is a binary relation that partitions a set into disjoint subsets. Thus, an equivalence relation can be used to partition a state set Q into the set of its *equivalence classes*. States that belong to the same equivalence class can be merged and consequently an FSM with less states can be obtained, which is referred to as the *quotient FSM*. The quotient FSM is an abstracted FSM. After merging states, the new state has the union of incoming and outgoing transitions of the merged states. The formal definition of quotient state machine is given in Paper 2 and 3 in Defs. 4 and 5, respectively.

Note that, in Paper 1, an EFSM is first transformed to an FSM before applying state merging abstraction.Thus, in the following the definitions are given for FSMs.

Two fundamental equivalence relations that play an important role in this thesis are *bisimulation equivalence* and *observation equivalence*.

Bisimulation requires two equivalent states to have the same future behaviour. Thus, the formal definition of bisimulation [14] is based on relation on states.

Definition 3 *[14] Let $G = \langle \Sigma, Q, \rightarrow, Q^{\circ}, Q^{\omega} \rangle$ be an FSM. An equivalence relation $\approx \subseteq Q \times Q$ is called a* bisimulation *on G, if the following holds for all $x_1, x_2 \in Q$ such that $x_1 \approx x_2$: if $x_1 \xrightarrow{\sigma} y_1$ for some $\sigma \in (\Sigma_{\omega} \cup \Sigma_{\tau})$, then there exists $y_2 \in Q$ such that $x_2 \xrightarrow{\sigma} y_2$ and $y_1 \approx y_2$.*

Bisimulation considers states to be equivalent if they have the outgoing transitions with the same events including the silent and marking events to equivalent states. Bisimulation can be computed by an efficient partition refinement algorithm [15]. This algorithm represents an equivalence relation as a partition, i.e., a set of equivalence classes each representing a set of equivalent states. The algorithm starts with an initial partition consisting of only one equivalence class contains all the states of an FSM, which is iteratively refined until a stable partitioning is reached. At each step, those states in equivalence classes that do not transit to the same equivalence classes on the same event are separated into other equivalence classes. This efficient algorithm gives the *minimal* FSM $\tilde{G}$, which is bisimilar to the original FSM G.

It is possible to relax bisimulation by ignoring the silent events. Then we can consider two states equivalent if from both of them equivalent states can be reached by the same sequences of events aside from silent events. This results in *weak bisimulation*, also known as *observation equivalence*. In order to ignore

the silent events, *natural projection* $P\colon \Sigma_\tau^* \to \Sigma^*$ is used which removes silent events τ from every string s.

Definition 4 *[14] Let $G = \langle \Sigma, Q, \to, Q^\circ, Q^\omega \rangle$ be an FSM. An equivalence relation $\sim \subseteq Q \times Q$ is called an* observation equivalence *on G, if the following holds for all $x_1, x_2 \in Q$ such that $x_1 \sim x_2$: if $x_1 \xrightarrow{s_1} y_1$ for some $s_1 \in (\Sigma_\omega \cup \Sigma_\tau)^*$, then there exists $y_2 \in Q$ and $s_2 \in (\Sigma_\omega \cup \Sigma_\tau)^*$ such that $P(s_1) = P(s_2)$, $x_2 \xrightarrow{s_2} y_2$, and $y_1 \sim y_2$.*

For observation equivalence a generalised version of the bisimulation algorithm [15] can be used. The only difference is that a split is performed on each known equivalence class C, separating states x with $x \xrightarrow{\tau^* P(\sigma) \tau^*} C$, for all $\sigma \in (\Sigma_\omega \cup \Sigma_\tau)$, from other states, i.e, the *transitive closure* of the silent transitions needs to be considered. Similar to bisimulation the algorithm gives the minimal FSM.

Besides state merging abstraction, an FSM can be abstracted by removing redundant transitions. More precisely, a transition $x \xrightarrow{\sigma} y$ is *observation equivalence redundant* and can be removed [16] if the FSM G contains a *matching path*. A matching path starts from x and ends up in the state y by a string consisting of σ and sequences of silent events before or after σ. The matching path must not contain the transition itself. After removal of the redundant transitions from G the abstract FSM H is obtained. The following definition describes how G and H are related.

Definition 5 *Let $G = \langle \Sigma, Q, \to_G, Q^\circ \rangle$ be an FSM. FSM $H = \langle \Sigma, Q, \to_H, Q^\circ \rangle$ with $\to_H \subseteq \to_G$ is a result of* observation equivalence redundant transition removal *from G, if for all transitions $x \xrightarrow{\sigma}_G y$ there exist $x \xrightarrow{\tau^* P(\sigma) \tau^*}_H y$.*

Bisimulation and observation equivalence are well-known general abstraction methods to merge states. In Chapter 4, bisimulation and a restricted version of observation equivalence are used to abstract FSMs.

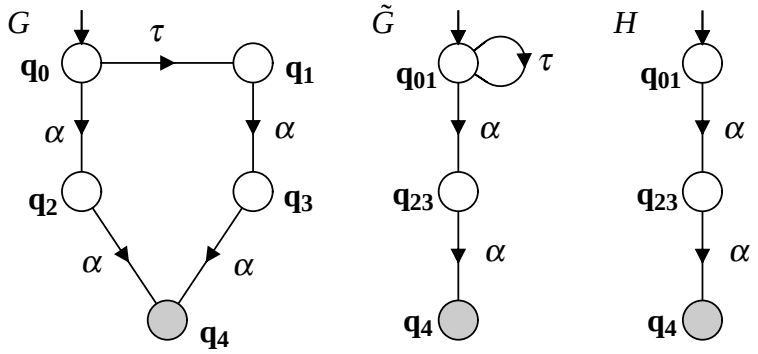


Figure 2.6: Example of observation equivalence based abstractions.

Example 4 *Consider the FSM G shown in Fig. 2.6. States q_2 and q_3 are bisimilar as state q_4 can be reached from both of them by executing event α. Moreover, states q_0 and q_1 can be merged by applying observation equivalence. These two state merging abstraction steps result in the abstracted FSM $\tilde{G}$ shown in Fig. 2.6. Next, transition $q_{01} \xrightarrow{\tau} q_{01}$ is redundant because of $q_{01} \xrightarrow{\varepsilon} q_{01}$, where ε is the empty string, and can be removed, resulting in the abstracted FSM H in Fig. 2.6.*

Chapter 3

Supervisory Control Theory

A discrete event system usually consists of a set of *plants* and *specifications* modelled as interacting state machines. Plants 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 specified requirements, for example, a machine trying to add a workpiece in a buffer that is currently full. Thus, commonly for a system a set of specifications is defined that describe the desired behaviour of the system. Now the task is to first *verify* whether the system satisfies the given specification, and, if not, restrict the system behaviour such that the given specification is fulfilled.

The *supervisory control theory* [1] provides a mathematical framework to automatically calculate, or *synthesise*, a control function called a *supervisor* that restricts the behaviour of the plant such that the specification is always fulfilled. In this thesis, plants, specifications, and supervisors are usually denoted by G, K, and S, respectively.

Fig. 3.1, shows the feedback loop of supervisor and plant. The plant generates events in Σ and the supervisor as a function $S(.)$, based on the earlier generated events, influences the plant behaviour, and thus the closed-loop system, by deciding whether or not to enable the possible events. Thus, the supervisor itself is incapable of generating events and only enables or disables them. In [17],

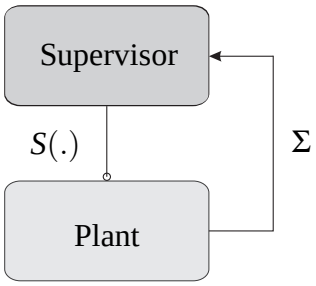


Figure 3.1: The feedback loop of supervisor and plant.

it was shown that when the plant and supervisor are modelled as FSMs, synchronous composition of the plant and supervisor can describe the behaviour of the plant under the control of the supervisor. This results in a simplified notion of controlled behaviour.

Supervisory control theory is generalised for nondeterministic models in [18–20] among others. In [18, 19], even though the plant may be nondeterministic, the specification must be deterministic. This condition is relaxed in [20], where both the plant and specification can be nondeterministic, with the objective that the controlled system be bisimulation equivalent to the specification. This thesis considers systems where both plants and specifications are modelled by deterministic finite-state machines, and the nondeterminism considered in this thesis is the result of abstraction.

3.1 Requirements for Supervisors

A plant describes everything that the uncontrolled system is capable of doing and the specification expresses the desired behaviour. Then a supervisor is a device that restricts the plant behaviour such that the plant in the closed-loop with the supervisor acts as desired. Besides this essential requirement, there are three more requirements that a supervisor should have.

3.1.1 Nonblocking

The supervisor is designed to fulfil a given specification. However, this is not per se useful if the supervisor restricts the plant from doing what it is supposed to do, for example, if the plant under the control of a supervisor gets stuck in a state or a loop from which no tasks can be completed. To avoid these kinds of situations, as mentioned in Chapter 2, some states of particular interest in the plant and the specification can be marked. Then the idea is to design a supervisor such that the closed-loop system can always reach a state that is marked by both the plant and the specification. Such a supervisor is referred to as a *nonblocking* supervisor [1].

Definition 6 *Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$. A state $x \in Q$ is called* reachable *in G if $Q^\circ \xrightarrow{s} x$ for some $s \in \Sigma_\tau^*$, and* coreachable *if $x \xrightarrow{t} Q^\omega$ for some $t \in \Sigma_\tau^*$. G is said to be* nonblocking *if every reachable state is coreachable.*

An FSM is nonblocking if from all the reachable states a marked state can be reached by executing a sequence of events. Given a plant G and a supervisor S the resultant closed-loop behaviour is G/S (reading as S controlling G), and the closed-loop system should be nonblocking. The nonblocking definition can be

easily extended to an EFSM. An EFSM is nonblocking if the resultant flattened FSM is nonblocking. The formal definition of nonblocking for EFSM is given in Paper 1 in Def. 11. As mentioned in Section 2.3 in this thesis, the special event ω is used to represent the marking of states. Then, the nonblocking definition should also be adapted to event-based marking.

Definition 7 *[21] An FSM $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ is* nonblocking, *if for every state $x \in Q$ and every trace $s \in \Sigma_\tau^*$ such that $Q^\circ \xrightarrow{s} x$ there exists $t \in \Sigma_\tau^*$ such that $x \xrightarrow{t\omega}$.*

Similarly to Def. 6, Def. 7 says that if a state is reachable from an initial state by a sequence of events from Σ_τ, then there must be from that state a sequence of events that ends with ω. In Def. 7, the condition that s and t are constructed from the set Σ_τ and thus do not contain ω is important. For example, if $s \in \Sigma_{\tau,\omega}^*$ then the state $\perp$ that is used to represent the marking of a state becomes reachable. As no ω transition is coming out of $\perp$, the FSM may falsely seem blocking.

3.1.2 Controllability

A supervisor is a device that restricts the plant behaviour by disabling some events. It is reasonable to assume that some events cannot be disabled by the supervisor, for example, the breaking of a device. Thus, 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 cannot. Now the question arises whether the silent event τ and the termination event ω are controllable or uncontrollable. The termination event is considered as a controllable event because the supervisor should be able to disable ω in order to remove some markings. Moreover, in this thesis having a silent event in the model of a system is the result of hiding a local event. Thus, in the case that the controllability of events is relevant, the silent event will have the same controllability characteristic as the local event that it replaces. In that, the notation of τ_u and τ_c is used for the uncontrollable and the controllable silent events, respectively. In the case that the controllability of the events are not relevant, the silent event τ is used, for example in Paper 1, where only nonblocking verification is considered.

To distinguish controllable events and uncontrollable events in the figures, uncontrollable events are prefixed by an exclamation mark (!).

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. In other words, a supervisor is controllable with respect to a plant if the occurrence of an uncontrollable event does not lead to a string which is not acceptable by the supervisor [1]. The formal definition of controllability

for deterministic FSMs is given in Paper 2 in Def. 5, and for nondeterministic FSMs in Paper 3, Def. 7.

3.1.3 Least Restrictiveness

The purpose of a supervisor is to restrict a plant behaviour to fulfil a given specification. It is typically required of a supervisor to achieve some minimum functionality. To ensure this minimum functionality, in this thesis the *least restrictive* supervisor, which restricts the system as little as possible is considered. Least restrictiveness may not always be required, and it can in fact ease the synthesis work. However, if a non-least restrictive algorithm synthesizes an overly restrictive supervisor, such as one giving an empty closed-loop system, it is not clear if this is due to a problematic plant/specification combination, or just an ill-chosen synthesis result. With an algorithm that guarantees a least restrictive supervisor, on the other hand, an overly restrictive supervisor will be the best achievable result and hence definitely a consequence of a problematic plant/specification combination. And of course, if you are going to do something why not try to do it optimally!

In this thesis, one goal is to calculate a least restrictive, controllable, and nonblocking supervisor and such a supervisor always exists and is unique with respect to a given plant and specification [1].

3.2 Synthesis and Verification

When we are dealing with safety critical systems such as medical devices or systems where errors are expensive such as factories, it is important to know if the systems works as expected in all possible situations. *Formal verification* is used to prove that a system satisfies a given specification. Two important properties that are typically verified are *nonblocking* and *controllability*. Nonblocking verification proves that a system can always complete a certain significant sub-task without violating the specification, and controllability verification proves that the system does not uncontrollably violate the specification.

In this thesis only, nonblocking verification of systems modelled as a set of EFSMs are considered. Nonblocking and controllability verification of FSM models are well-developed in the literature [1, 6, 7]. It was shown in [22] that controllability problems can be converted into nonblocking problems, making it possible to verify both nonblocking and controllability by running the nonblocking verification algorithm only once.

The straightforward approach to verify the nonblocking property is to compose all the components of the system. In the case that the system is modelled as a set of EFSMs, the next step is to flatten the composed EFSM. The final step

is to check if the resultant FSM is nonblocking by inspecting if it is possible to reach a marked state from all the reachable states.

Example 5 *Consider the manufacturing system in Example 1. The EFSM model of the system is shown in Fig. 2.3 and the synchronous composition result is shown in Fig. 2.5. To use the straightforward approach to verify the nonblocking property of the system, the system needs to be flattened first. Fig. 3.2 shows the flattened FSM H. Initially the buffer is empty, which means $b = 0$, and M_1 and M_2 are in initial locations I_1 and I_2. Thus, the initial state of H is $(0, I_1, I_2)$. At this state the event s_1 can occur. After occurrence of this event, the buffer is still empty and M_2 is still in the location I_2. M_1 however, moves to location W_1. Thus the next state of the system is $(0, W_1, I_2)$ and so on. The flattened FSM H is isomorphic to FSM $M_1 \parallel B \parallel M_2$ shown in Fig. 2.4. This confirms that both the FSM and EFSM models describe the same behaviour. FSM H is nonblocking as from all the states, the marked state $(0, I_1, I_2)$ can be reached. Thus, the original EFSM system is nonblocking.*

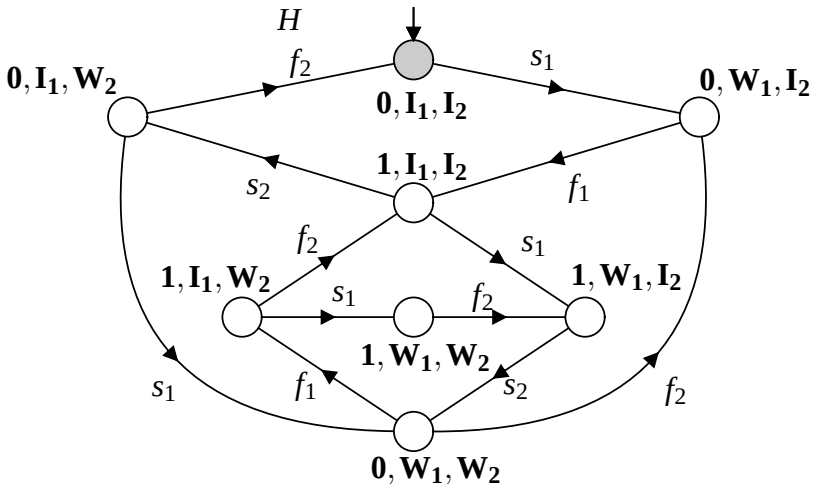


Figure 3.2: The flattened FSM of the EFSM model of the manufacturing system.

After verification, if the result is satisfactory then the task is done and there is no need to design a supervisor as the specification can be used as a supervisor. Otherwise, a supervisor needs to be designed to prevent the system to go to the bad states. To this end, supervisory control theory [1, 6], which automatically *synthesises* such a supervisor is proposed.

The synthesis algorithm in this thesis first transforms all the specifications to plants [22]. A specification FSM 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 $\perp$. The formal definition of *plantification* is given in Paper 2, Def. 8. Plantification essentially transforms all initial controllability problems into blocking problems. Then, the plant G and the transformed specification $K^{\perp}$ are synchronised and the synthesis algorithm iteratively identifies and removes blocking states, and the states that uncontrollably go to blocking states. The algorithm in the end returns the least restrictive nonblocking and controllable behaviour allowed by a specification K with respect to a plant G.

Example 6 *Consider the manufacturing system in Example 1. The events* $!f_1$ *and* $!f_2$ *are uncontrollable and* s_1 *and* s_2 *are controllable. The safety issue in this system is that machine* M_1 *should not try to put a workpiece in the buffer B if the buffer is currently full, and* M_2 *should not try to remove a workpiece if the buffer is empty. Therefore, FSM B is considered as the specification to avoid buffer overflow and underflow and the machines are considered as the plants. This specification is uncontrollable as it disables the uncontrollable event* $!f_1$ *in the states* (F,W_1,W_2) *and* (F,W_1,I_2) *to avoid adding a workpiece in a currently full buffer. Thus, the plant violates the specification uncontrollably and a supervisor needs to be designed. The first step to design a supervisor is to plantify the specification. Fig. 3.3 shows the FSM* $B^{\perp}$*, which is the result of plantification. Next,* $B^{\perp}$ *is composed with* M_1 *and* M_2*. The state* $\perp$ *in the composed FSM* $B^{\perp} \parallel M_1 \parallel M_2$ *represents the initial controllability problem, and it is a blocking state as can be seen in Fig. 3.3. The plant needs to be restricted to avoid ending up in the blocking state* $\perp$*. This state however, is reached from* (F,W_1,W_2) *and* (F,W_1,I_2) *by the uncontrollable event* $!f_1$*, which the supervisor cannot disable. Thus, the best control decision is to disable event* s_1 *in the states* (F,I_1,W_2) *and* (F,I_1,I_2) *and thus avoid starting machine* M_1 *when the buffer is full. The least restrictive, nonblocking and controllable supervisor S is shown in Fig. 3.3.*

In the finite-state case, the iteration to remove all problematic states is guaranteed to terminate, and the complexity is $O(|Q||\rightarrow|)$, where $|Q|$ and $|\rightarrow|$ are the numbers of states and transitions of the state machine. As shown by [23] the synthesis problem is NP-hard, since the size of Q and $\rightarrow$ grows exponentially with the number of components. Thus, the straightforward approaches for verification and synthesis described in Section 3.2 are limited by the *state-space explosion* problem. Therefore, this thesis proposes a *compositional* approach, described in Chapter 4, to solve synthesis and verification problems more efficiently.

3.3 Problems Considered

We assume that the components of a system are given as *deterministic* finite-state machines or extended finite-state machines. The individual components

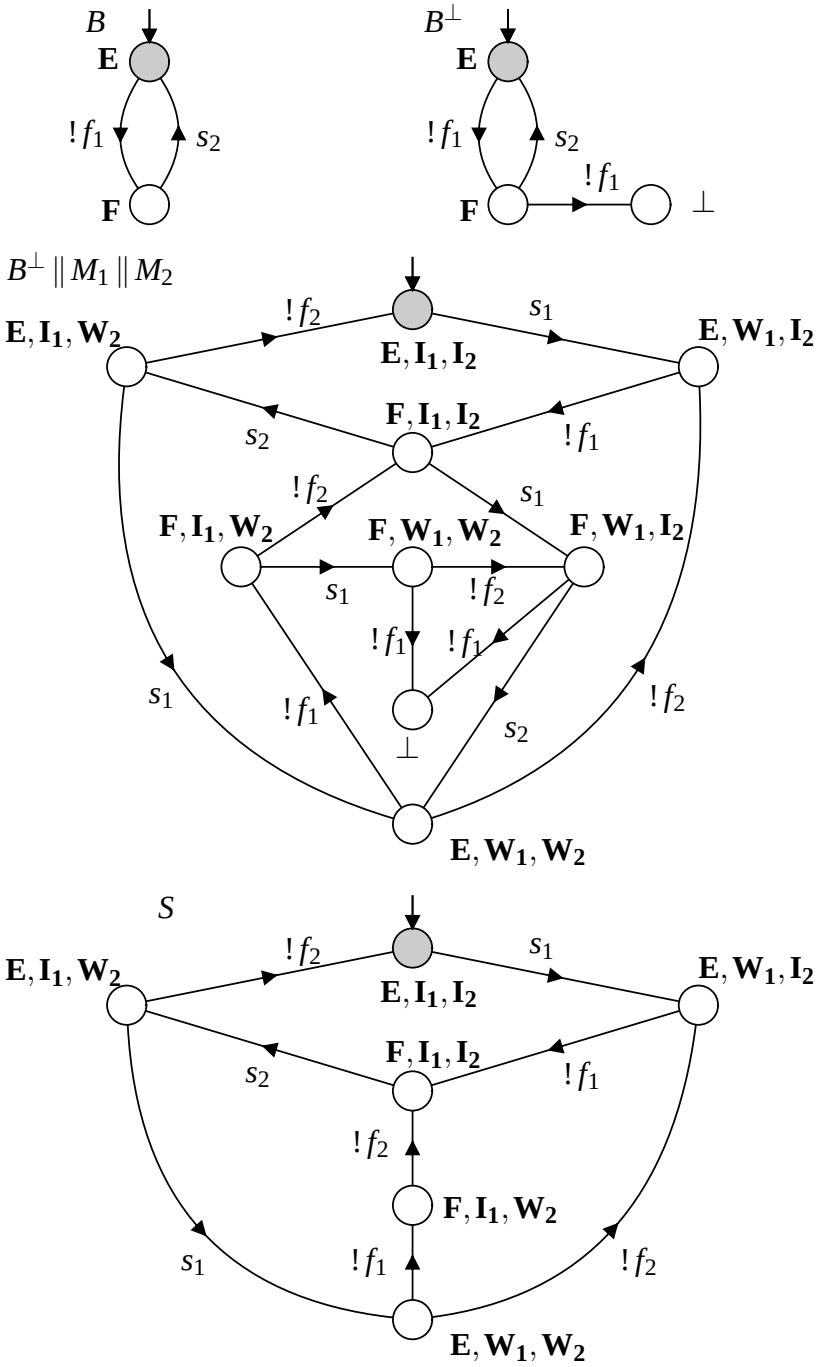


Figure 3.3: The specification and the supervisor of the manufacturing system.

have arbitrary alphabets, all the events of the systems are observable and the controllability characteristic of an event does not change from one component to another.

To be more detailed, the two problems considered are:

The modular nonblocking verification problem A modular system is modelled as

$$\mathscr{E} = \{E_1, \ldots, E_n\} \tag{3.1}$$

where each E_i is an extended finite-state machine. The task is to verify whether the system $\mathscr{E}$ is nonblocking. As only nonblocking verification is considered, the controllability of the events is irrelevant. Also, there is no need to treat specifications and supervisors differently from plants.

The modular supervisor synthesis problem A modular plant $G = \{G_1, \ldots, G_m\}$ and a modular specification $K = \{K_1, \ldots, K_l\}$ are given as FSM models. As mentioned before, the specifications are plantified and thus, the system is

$$\mathscr{G} = \{G_1, \ldots, G_n\} \tag{3.2}$$

The task is to calculate a least restrictive, controllable and nonblocking supervisor which has a modular structure. Here, as the supervisor not only needs to be nonblocking but also controllable, in contrast to nonblocking verification, the controllability of events is considered.

Chapter 4

The Compositional Approach

Usually discrete event systems are *modular* in the sense that the model of the system consists of a set of plant components and a set of specifications, all interacting with each other. The straightforward way to analyse a system involves building an explicit monolithic model which may lead to the inherent complexity problem known as the *state-space explosion* problem. This combinatorial problem occurs as the number of states grows exponentially with the number of components. This problem makes it intractable to examine the monolithic state-space of a system due to lack of memory and time. Consequently, constructing the explicit monolithic model of the system is not efficient and methods to exploit the modular structure or methods that efficiently represent the state-space of the system are needed. One way to exploit the modular structure of a system is to use a *compositional approach*. The compositional approach has previously been successfully used for *verification* of discrete event systems [2, 24–27]. In this thesis, the compositional approach is used both for verification and synthesis of systems, which are respectively modelled as a set of interacting EFSMs and FSMs.

In this chapter, Section 4.1 briefly overviews different existing approaches that help to avoid the state-space explosion problem. The general compositional approach is described in Section 4.2. The compositional nonblocking verification algorithm for EFSM systems is explained in Section 4.3. Finally, Section 4.4 describes two frameworks for compositional synthesis of FSM.

4.1 Alleviating the State-Space Explosion Problem

Various approaches to avoid state-space explosion have been proposed in the literature. This section briefly describes the *modular*, *hierarchical*, and *symbolic representation* approaches.

The modular approach was first introduced in [28]. In this work it was shown

that controllability verification and controllability synthesis can be done by considering one specification at a time with the plant components it imposes requirements on. This approach was later developed in [29, 30]. The modular approach is very efficient, however all the works mentioned only consider controllability of a supervisor, and they do not guarantee global nonblocking of the closed-loop behaviour. The work in [31], resolves conflict among modular supervisors. However, in this work the supervisor is not necessarily least restrictive.

Hierarchical approaches divide the system into different levels of hierarchy. This was first introduced by [32], and was later developed in [33]. In [33], the authors divide the system into high-level and low-level subsystems where subsystems communicate through *interfaces*. More recently, decentralized and hierarchical approaches are presented in [34–36]. In these works, the authors obtain decentralized supervisors for each specification and partition the plant components and decentralized supervisor into subsystems. *Natural projection* with the observer property is used as a method to abstract each subsystem. The work in [34] does not necessarily result in a least restrictive supervisor. To guarantee least restrictiveness, *output control consistency* [35] and the less restrictive condition of *local control consistency* [36] are proposed.

Algorithms based on Binary Decision Diagrams (BDDs) [37] convert the model of the system to a symbolic representation in the form of BDDs [38] and explore the full state-space symbolically [39]. By symbolic we mean that during analysis, the system is not represented directly as states and transitions but indirectly as Boolean functions. Representing a system symbolically, in many cases results in smaller representation of the state-space of the system. BDDs were first brought into the supervisory control theory by [40] and were later developed by [41–43].

4.2 General Compositional Approach

In this thesis, the compositional approach is used to alleviate the state-space explosion problem, and it is used both for nonblocking verification and supervisor synthesis. If the task is to verify nonblocking, then the controllability of the events is irrelevant and thus, the specifications can be considered as plants. Otherwise, the specifications are first plantified as explained in Section 3.2. Therefore, in both nonblocking verification and synthesis, the input to the compositional approach is a set of interacting plant components

$$\mathscr{G} = G_1 \parallel \cdots \parallel G_n. \tag{4.1}$$

To alleviate the state-space explosion problem, the compositional approach constructs the monolithic model gradually, while abstracting components at each

step. Before beginning the synchronisation process, each individual component is first abstracted, and the abstractions replace the original components. By abstractions we mean removing redundancy, and an abstracted component has less states or transitions compared to the original component. If no more abstraction is possible then some components need to be composed or, if the system is an EFSM system, some variables are removed by *partially unfolding* them (as explained in Section 4.3.2). Iteratively, the intermediate results are composed and abstracted again and again, until eventually, the procedure leads to a single state machine, which is an abstract representation of the system. This state machine has less states and transitions than the original system. The final step is to use the final abstracted state machine for either verification or supervisor calculation. Fig. 4.1 illustrates the general compositional approach.

The compositional approach explained above is general and can be used for any system and any property of interest. Once the property of interest is determined, we can define abstraction methods that abstract each component in such a way that the property is preserved. Abstraction methods play an important role in the efficiency of the compositional algorithm. The reason is that the size of the final component depends on the abstractions at each step and the smaller the size, the more efficient the analysis.

Another main ingredient for the compositional approach to work is heuristics, which decide what components to compose at each step of the algorithm.

In the following sections, some general points regarding hiding events, which is essential for abstraction, heuristics, and issues that need to be considered when abstracting components are discussed.

4.2.1 Local Events and Hiding

The state-space explosion problem is more noticeable when the components are loosely coupled, which means some components have internal behaviours independent of others. While this independence can result in state-space explosion, it can also be useful when abstracting components in the compositional approach.

The events that represent the internal behaviour of components are referred to as *local events*. The reason that they are called local is that these events appear locally in only one component of the system. In this thesis, the set Υ denotes the set of local events. Non-local, *shared*, events are denoted by $\Omega = \Sigma \setminus \Upsilon$. In general, abstraction methods depend on the local events and the more local events, the more possibility of abstraction.

In compositional nonblocking verification, the identity of the local events can be hidden. Local events can thus be replaced by the silent event τ. This is possible because hiding of local events does not change the nonblocking property of a system [2]. Paper 1 only considers compositional nonblocking verification and

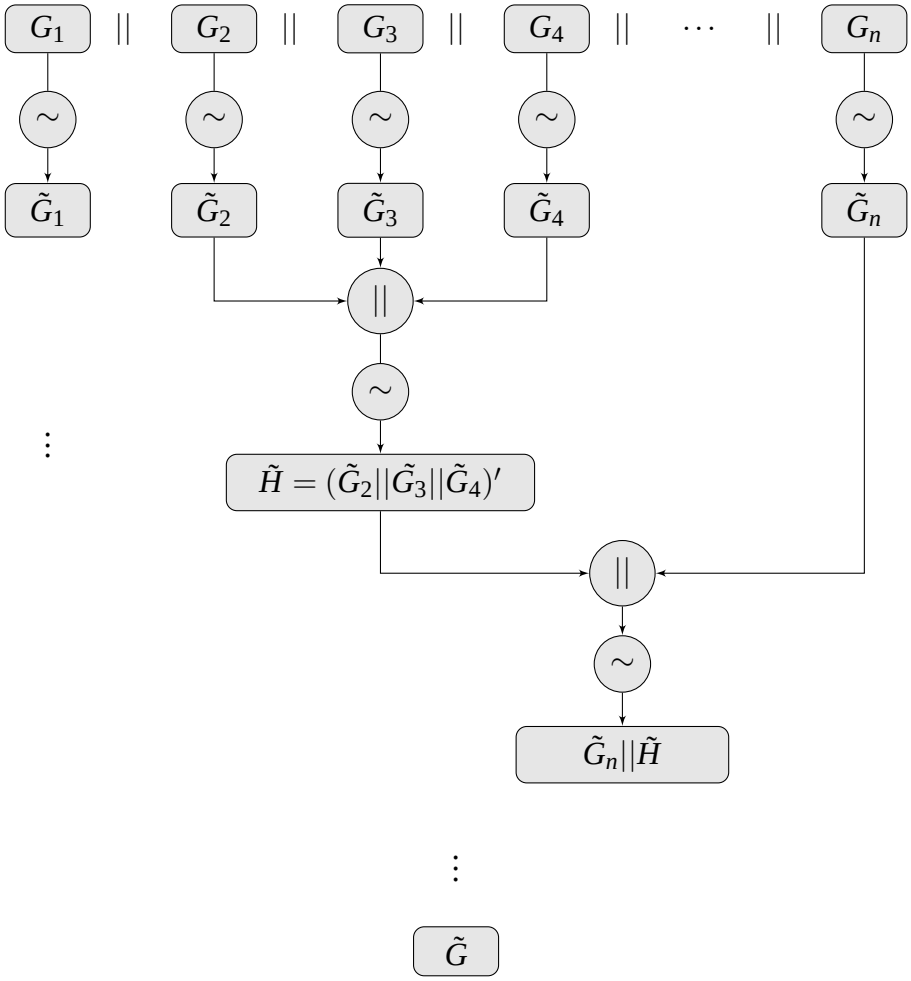


Figure 4.1: General compositional approach. The modular system is described by $\{G_1, G_2, \cdots, G_n\}$ which is a set of plant state machines and $\sim$ is a proper equivalence relation.

thus, the local events are hidden when possible. In the compositional synthesis approach presented in Paper 2, the local events are not hidden as the supervisor may need to know the identity of the events to make control decisions. Paper 3, similar to Paper 2, uses the compositional approach for supervisor calculation. However, as the supervisor in Paper 3 makes control decisions based on the states rather than the events, hiding is possible in this paper. In this paper the local uncontrollable and controllable events are replaced by τ_u and τ_c, respectively.

In the figures, if local events are not hidden, they are shown with parentheses around them.

4.2.2 Abstraction Methods

Generally, the compositional approach attempts to replace individual components by abstracted versions. This requires that the abstracted components are properly 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.

For compositional nonblocking verification, the property of interest is nonblocking. Thus, we consider two state machines equivalent if both have the same nonblocking results in synchronisation with an arbitrary test state machine. This equivalence relation is called *conflict equivalence*. It was introduced in [44] and is used in Paper 1. The test state machine essentially represents the rest of the system and the reason for considering it arbitrary is to have a general framework, which works for arbitrary systems.

In the case of compositional synthesis, the intention is 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 a supervisor $\tilde{S}$ by the compositional approach, the equivalence relation could be defined as either maintaining the same supervisor as the monolithic supervisor, $\mathscr{L}(S) = \mathscr{L}(\tilde{S})$, or having the same closed-loop behaviour $\mathscr{L}(G \parallel S) = \mathscr{L}(G \parallel \tilde{S})$. Technically, supervisors are calculated to modify the closed-loop behaviour of the system such that the specification is fulfilled. Consequently, in Paper 2 and 3, maintaining the same closed-loop behaviour is considered as the property of interest when calculating a supervisor. This equivalence relation is called *synthesis equivalence*.

4.2.3 Heuristics

The efficiency of the compositional algorithms is sensitive to the order in which state machines are composed and abstracted. As there are many options at each step, a number of heuristics has been defined to decide what state machines to compose.

As mentioned before, abstraction methods play an important role in the performance of the compositional approach, and usually local events and hiding are essential for abstraction. Moreover, it is important to keep the intermediate results small. Based on these principles a variety of heuristics to decide what FSMs of a system to compose are proposed in [2]. These heuristics are used in the compositional synthesis approach in Paper 2 and 3.

For compositional nonblocking verification of EFSM systems, the algorithm, besides composing EFSMs at each step, also needs to gradually remove variables. Thus, the heuristics for EFSM systems do not only consider what EFSMs to compose, but also which variables to remove at each step. In general, it is a good idea to remove variables with small domains as this produces smaller intermediate results, or variables that appear frequently as removing them simplifies large numbers of updates. Different heuristics for EFSM systems are proposed in Paper 1.

4.3 Compositional Verification of EFSM Systems

The compositional nonblocking verification algorithm for EFSM systems seeks to repeatedly apply conflict equivalence abstractions to individual EFSMs and partially unfold variables. In the end of the compositional algorithm, all the variables are partially unfolded and the system is simple enough to be verified monolithically.

As mentioned before, one of the most important steps in the compositional approach is abstraction. In [2], a variety of abstraction methods for FSMs are proposed that preserve conflict equivalence. This framework is extended to EFSM systems in Paper 1. In general, the nonblocking property of an EFSM system is not necessarily preserved after applying the abstraction methods defined for FSMs to an EFSM without considering the updates of the EFSM. In the following, by the help of an example it is shown how updates in an EFSM affect the abstraction methods defined for FSMs.

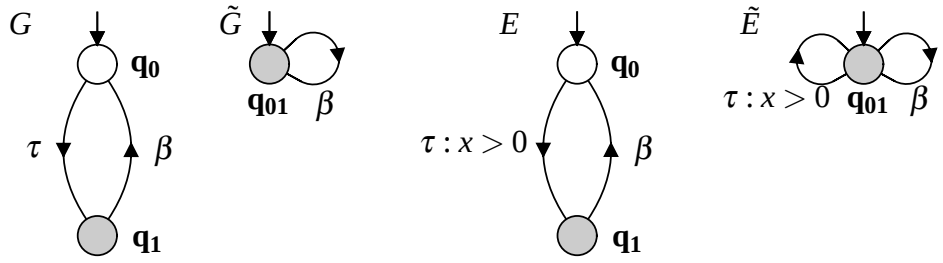


Figure 4.2: FSM $\tilde{G}$ and G are conflict equivalent, while EFSM $\tilde{E}$ and E are not conflict equivalent.

Example 7 *Consider the FSM G and EFSM E in Fig. 4.2. The events of the transitions $q_0 \xrightarrow{\tau} q_1$ in G and E are local and hidden. First consider the FSM G. Clearly from q_0 and q_1 the same states can be reached by executing the same events if the silent event τ is not considered. Thus, q_0 and q_1 are observation equivalent, Def. 4. As observation equivalence preserves conflict equivalence [45], states q_0 and q_1 can be merged, resulting in the conflict equivalent FSM $\tilde{G}$ shown in Fig. 4.2.*

Now consider the EFSM E. The domain of the variable x is $\{0,1,2\}$ and $x^\circ = 0$. This EFSM is blocking because, to reach the marked location q_1, the transition $q_0 \xrightarrow{\tau:x>0} q_1$ needs to happen, and as initially $x = 0$, this is impossible. If we apply to E the same abstraction method as on G, without considering the update $x > 0$, the EFSM $\tilde{E}$, shown in Fig. 4.2, is obtained, which is clearly nonblocking. *Thus, E and $\tilde{E}$ are not conflict equivalent.*

In Example 7, we blindly applied the same abstraction method on both G and E. However, since the update of the silent event in E was disregarded, E and $\tilde{E}$ in contrast to G and $\tilde{G}$ are not conflict equivalent.

Now questions arise. Can we apply conflict equivalence abstractions developed for FSM on EFSMs? When can we use a silent event for an abstraction? How can we remove variables from the system without facing the state-space explosion problem?

All the above questions are answered in Paper 1, and some details of the paper are discussed in the following sections.

4.3.1 Normalisation

At first sight, it seems that the main obstacle to apply the abstraction methods developed for FSMs to EFSMs is the updates of EFSMs. In an EFSM system, transitions of components are labelled by events *and* updates, and most likely each event associates with different updates in different transitions, as this gives the user more freedom to model the system. This makes it hard to see how executing an event in an EFSM system affects the variables. The first step of the compositional nonblocking verification algorithm is therefore *normalisation*. Normalisation associates each event with its own distinct update. To normalise an EFSM system, components are first individually normalised and then the system is globally normalised.

A component is not normalised if an event appears on different transitions with different updates. To individually normalise components we use *renaming*, which introduces new events to the system for each update p associated with event σ. Note that the events introduced by renaming should not already be in the alphabet of the system, otherwise the renaming process may not converge. After renaming one component, the other components of the system are changed to use

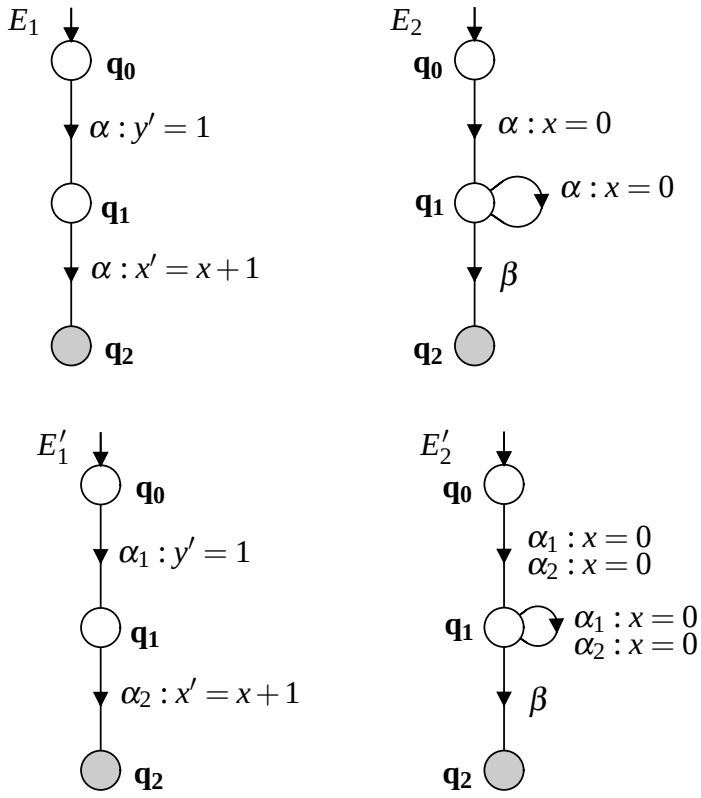


Figure 4.3: The result of individual normalisation of E_1 and E_2.

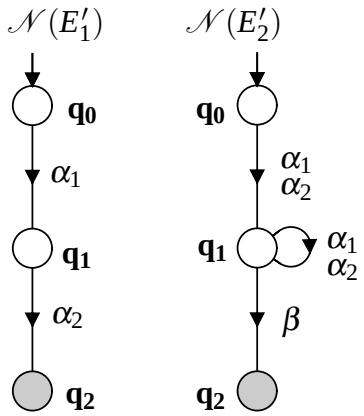
the new events. Prop. 2 in Paper 1 confirms that the behaviour of a system before and after individual normalisation of each component is identical up to renaming of the events. As renaming preserves the nonblocking property, normalisation of individual EFSMs does not change the nonblocking property of the system.

Example 8 *Consider the EFSM system $\{E_1, E_2\}$, where E_1 and E_2 are shown in Fig. 4.3. EFSM E_2 is already individually normalised as both α and β in this EFSM have unique updates. However, EFSM E_1 is not normalised as the event α appears with updates $y' = 1$ and $x' = x+1$. To normalise EFSM E_1, the events α_1 and α_2 are introduced. These events replace the event α in transitions $q_0 \xrightarrow{\alpha:y'=1} q_1$ and $q_1 \xrightarrow{\alpha:x'=x+1} q_2$ of E_1 respectively, which results in the normalised E_1' shown in Fig. 4.3 bottom left. Now, we need to replace the event α in E_2 by the new events to comply with the event modification. The updates of the events α_1 and α_2 in the modified EFSM E_2' are equal to the update of α in E_2. The normalised EFSM E_2' is shown in Fig. 4.3 bottom right.*

After normalising individual components, the system needs to be globally normalised. This is done by assigning to each event an update, which is the conjunction of all the updates associated with that event in the EFSMs in the system.

The updates associated to each event after this step are essentially the updates that would have been calculated by synchronisation. However, normalisation in contrast to synchronisation retains the modular structure, which is essential for the compositional approach.

If an EFSM system is normalised, the need to have updates on each transition is removed. Moreover, the synchronisation task becomes simpler because updates can be disregarded and standard FSM synchronisation can be used for normalised EFSM systems.



Event	Update
α_1	$y' = 1 \wedge x = 0$
α_2	$x' = x+1 \wedge x = 0$
β	*true*

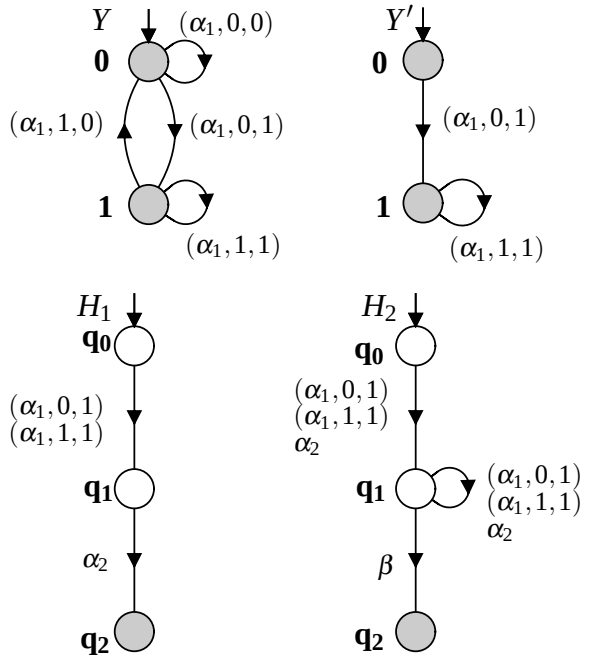
Figure 4.4: The normalised system in Example 9.

Example 9 *Consider the EFSM system $\{E_1', E_2'\}$ shown in Fig. 4.3 where components are individually normalised. The update of the event α_1 is $y' = 1$ in E_1' and $x = 0$ in E_2'. To globally normalise the system, the update of α_1 becomes the conjunction of $y' = 1$ and $x = 0$, which is $y' = 1 \wedge x = 0$. Similarly, the update of α_2 is $x' = x+1$ in E_1' and $x = 0$ in E_2', and thus the update of α_2 in the normalised system is $x' = x+1 \wedge x = 0$. The update of the event β does not change as this event is local to E_2'. After normalisation of the system, writing the updates on the transitions becomes unnecessary, and the information regarding the update of each event is given in the table in Fig. 4.4. The figure also shows $\mathscr{N}(E_1')$ and $\mathscr{N}(E_2')$, which replace E_1' and E_2' respectively after the normalisation procedure.*

The normalisation procedure preserves the nonblocking property of the system and is explained in detail in Section 4 of Paper 1. From now on we assume that EFSM systems are normalised.

4.3.2 Partial Unfolding

As mentioned before, the straightforward approach to verify an EFSM system unfolds all the variables of the system at once, which results in the state-space explosion problem. To alleviate this problem, the compositional approach for



Event	Update
α_1	$y' = 1 \wedge x = 0$
α_2	$x' = x+1 \wedge x = 0$
β	*true*
$(\alpha_1,0,0)$	*false*
$(\alpha_1,0,1)$	$x = 0$
$(\alpha_1,1,0)$	*false*
$(\alpha_1,1,1)$	$x = 0$
α_2	$x' = x+1 \wedge x = 0$
β	*true*

Figure 4.5: Example of partially unfolding a variable *y*.

EFSM systems unfolds variables gradually and replaces them by EFSMs called *variable EFSMs*. This process is referred to as *partial unfolding*.

Assume that we want to remove a variable v from the system and replace it with the variable EFSM V. The locations of the variable EFSM correspond to the domain of the variable v. To label the transitions of the variable EFSM, first events with the variable v in their updates are identified. These events, in combination with variable values create new events that label the transitions of the variable EFSM. An event of the form (σ, a, b) labels the transition $a \xrightarrow{(\sigma,a,b)} b$ of the variable EFSM V. Now, if the update of σ is p, the values a and b are substituted for v and v', respectively, in updates that have the variable v. This results in simpler updates with fewer variables, which are assigned as the updates of the new events (σ, a, b).

Since partial unfolding introduces new events in the system, all the EFSMs of the system need to be modified to use the new events.

Example 10 *Consider the normalised EFSM system shown in Fig. 4.4 with the updates in the table of the figure. Assume* $\mathrm{dom}(x) = \mathrm{dom}(y) = \{0,1\}$ *and* $x^\circ = y^\circ = 0$. *Partially unfolding the variable y results in the variable EFSM Y with locations 0 and 1 in Fig. 4.5 top left. The only event that has the variable y in its update is* α_1 *with update* $y' = 1 \wedge x = 0$. *This event is replaced by four new*

events representing (α, y, y') *for the possible combinations of y's domain values:*

$(\alpha_1, 0, 0)$ *with update* $(y' = 1 \wedge x = 0)[y \mapsto 0, y' \mapsto 0] \equiv 0 = 1 \wedge x = 0 \Leftrightarrow$ false
$(\alpha_1, 0, 1)$ *with update* $(y' = 1 \wedge x = 0)[y \mapsto 0, y' \mapsto 1] \equiv 1 = 1 \wedge x = 0 \Leftrightarrow x = 0$
$(\alpha_1, 1, 0)$ *with update* $(y' = 1 \wedge x = 0)[y \mapsto 1, y' \mapsto 0] \equiv 0 = 1 \wedge x = 0 \Leftrightarrow$ false
$(\alpha_1, 1, 1)$ *with update* $(y' = 1 \wedge x = 0)[y \mapsto 1, y' \mapsto 1] \equiv 1 = 1 \wedge x = 0 \Leftrightarrow x = 0$

The new events and their simplified updates are shown in the table in Fig. 4.5. All the events with false *update can be removed as the transitions labelled with these events can never happen. Fig. 4.5 shows EFSM* Y'*, which is obtained by removing transitions with* false *updates from* Y*. After introducing new events, the EFSMs need to be changed to use the new events. Fig. 4.5 shows* H_1 *and* H_2 *at the bottom, which replace* $\mathscr{N}(E_1')$ *and* $\mathscr{N}(E_2')$ *in Fig. 4.4 respectively.*

Note that after unfolding a variable *v*, we replace the variable in an update with different values. Then the system is not necessarily normalised if we do not introduce the new events in the system. Therefore, we replace σ with (σ, a, b).

The question at this point is how partial unfolding can help with the state-space explosion problem? Partial unfolding simplifies updates in the sense that they have less variables. This helps with the abstraction methods, as explained in the next section.

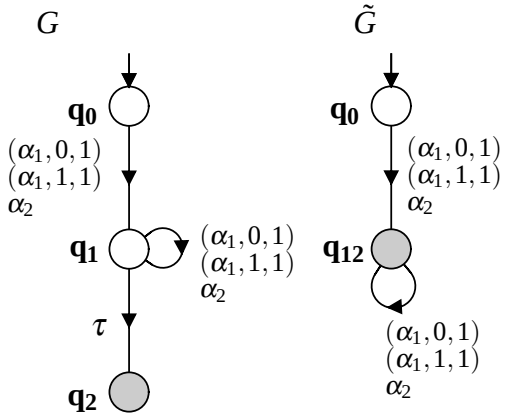
4.3.3 Adapting FSM Abstraction Methods for EFSMs

One of the contributions of Paper 1 is to find a way to apply the conflict equivalence abstraction methods defined for FSMs [2] directly on EFSMs. As most abstraction methods defined for FSMs use the silent event τ, the first step is to extend the notion of silent events into the EFSM framework.

Silent events replace *local* events, i.e., events that only appear in one FSM in the system. Local events can be replaced by silent events because they do not interact with other components of the system. In an EFSM system however, as shown in Example 7, only considering local events when abstracting an EFSM does not necessarily yield conflict equivalence abstraction. This is because even though local events do not interact with other components, the variables in the updates associated with those local events may be shared, and thus interact, with other components. Thus, intuitively we can hide events that are local and have no variables in their updates. As updates are predicates, updates with no variables can be replaced by *true* or *false*. As transitions with *false* update are always disabled, they can be removed from the system. This leaves us with *true* updates as the updates with no variables. Thus, we can hide events that are local and have *true* updates. This makes sense because transitions of an EFSM that are labelled

by silent events and *true* updates are always enabled, similarly to transitions labelled by silent events in an FSM.

Once we identified the local events that can be hidden, we can regard the normalised EFSM as an FSM, and hence apply all the abstractions defined for FSMs in [2]. This approach is explained in detail in Section 5.5 of Paper 1 and the proof of correctness is given in Section B.5 of Paper 1.



Event	Update
$(\alpha_1,0,1)$	$x=0$
$(\alpha_1,1,1)$	$x=0$
α_2	$x'=x+1 \wedge x=0$
β	*true*

Figure 4.6: Example of FSM-based conflict equivalence abstraction.

Example 11 *Consider the EFSM system H_1,H_2,Y' shown in Fig. 4.5. Event β is a local event with* true *update and thus, this event can be hidden. The next step is to consider the EFSM H_2 as an FSM and hide event β. Fig. 4.6 on the left shows the resultant FSM G. Now applying to G observation equivalence abstraction, which preserves conflict equivalence [2], results in merging of states q_1 and q_2 and the abstracted FSM $\tilde{G}$. Afterwards $\tilde{G}$ is regarded as an EFSM, by considering the updates again. $\tilde{G}$ is shown in Fig. 4.6 to the right.*

4.3.4 Experimental Results

The compositional verification has been implemented in the DES software tool Supremica [46] and applied on several large industrial models taken from [47–54]. The test cases include complex industrial models such as manufacturing systems and automotive body electronics. For a detailed discussion, we refer to Paper 1.

To evaluate the efficiency of the EFSM-based compositional nonblocking verification algorithm, its performance is compared with the BDD-based algorithm and FSM-based compositional algorithm both implemented in Supremica. The BDD-based algorithm converts an EFSM system to a symbolic representation and the FSM-based algorithm converts the EFSM system to a modular FSM system. Paper 1 explains in detail how an EFSM system is converted to an FSM system.

The compositional nonblocking verification algorithm successfully verifies all the test cases in a few seconds or minutes, while the BDD-based algorithm fails for large scaled-up systems and the FSM-based algorithm fails for EFSM systems with complicated updates. The reason that the BDD-based algorithm fails for some examples is, while the algorithm copes well with complicated updates it is limited by the size and search depth of the state-space, and the BDDs representing these systems are large. On the other hand, the reason for failure of the FSM-based algorithm for systems with complicated updates is that, the more complicated the updates in an EFSM system are, the more events appear in the converted FSM-based compositional algorithm. Thus, it takes a long time to convert the systems and in some cases, the conversion takes longer time than the verification itself.

The EFSM-based compositional algorithm outperforms the two well-developed verification algorithms in most of the cases, and shows promising results even for large industrial models.

4.4 Compositional Synthesis

This section discusses the compositional synthesis approach, which is the subject of Paper 2 and 3. In compositional synthesis, in contrast to verification, we are concerned with more than giving a “yes” or “no” answer, and the task is to remove the states that violate the specification. In addition, since we are interested in calculating a controllable supervisor, we need to take into account the controllability of the events.

As mentioned before, the property of interest in Paper 2 and 3 is synthesis equivalence, a property that preserves the same closed-loop behaviour.

In Paper 2 the supervisor is represented as a set of FSMs and in Paper 3 the framework is extended and the supervisor is a set of state maps. Having a supervisor as a state map allows nondeterminism after abstraction and transition removal abstraction, both of which are avoided in Paper 2.

In the following, some details regarding each approach are given.

4.4.1 State Machine-Based Supervisor

The supervisor calculated in Paper 2 is a set of FSMs that disables controllable events that the plant would otherwise have generated. As mentioned before, the property of interest in Paper 2 is synthesis equivalence, which requires the same closed-loop behaviour after each abstraction. In the framework of Paper 2, to preserve synthesis equivalence after each abstraction, the abstracted FSM needs to be deterministic. The following example makes it clear why this condition is essential.

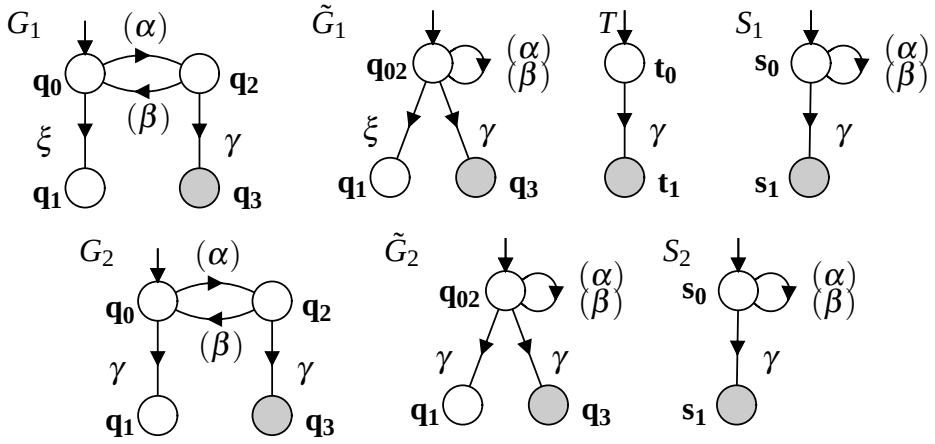


Figure 4.7: Abstraction of G_1 results in the deterministic FSM $\tilde{G}_1$, while abstraction of G_2 results in the nondeterministic FSM $\tilde{G}_2$.

Example 12 *Consider FSMs G_1, $\tilde{G}_1$, and T in Fig. 4.7. All events are controllable, and events α and β are local events in G_1. States q_0 and q_2 in G_1 can be merged because if from one of them a marked state can be reached, then from the other one a marked state can also be reached by executing the local events. Thus, synthesis always removes either both or none of them. This results in $\tilde{G}_1$, which is synthesis equivalent with G_1. Calculating a supervisor for $\tilde{G}_1$ and T results in S_1, shown in Fig. 4.7 to the right. Supervisor S_1 in composition with G_1 and T disables event ξ in state q_0 and enables event γ in q_2 of G_1. This supervisor will give the same closed-loop system when controlling G and T, as would the monolithic supervisor. Now consider FSM G_2 in Fig. 4.7. Applying the same abstraction method to G_2 results in $\tilde{G}_2$. The supervisor calculated from $\tilde{G}_2$ and T is S_2 shown in Fig. 4.7. This supervisor enables event γ in the states q_0 and q_2 of G_2. Thus, S_2 is a blocking supervisor for G_2 and T, since it permits the blocking state q_1 of G_2 to be reached.*

In Example 12, a correct supervisor needs to be aware of the states of G_2 in order to decide whether to enable the controllable event γ or not, and it is not straightforward to construct such a supervisor only from the abstraction $\tilde{G}_2$. For a supervisor FSM to work correctly, nondeterminism should be avoided after abstraction. Then we have two solutions: either we do not merge states if it leads to nondeterminism, which may make some desirable abstraction impossible, or we use renaming. In Paper 2, to avoid nondeterminism after abstraction, the idea of *distinguishing sensors* [55] is adapted and *renaming* is proposed. Similarly to renaming in Paper 1, renaming in the framework of Paper 2 introduces new events. After applying a renaming to one component, new events are introduced, so the remaining components need to be modified to use the new events.

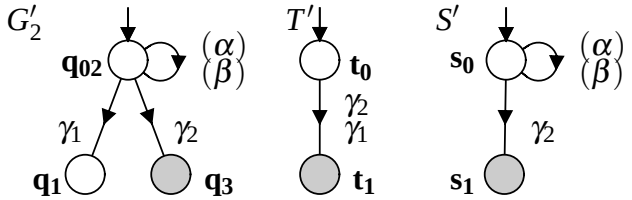


Figure 4.8: Applying renaming to G_2' to avoid nondeterminism after abstraction.

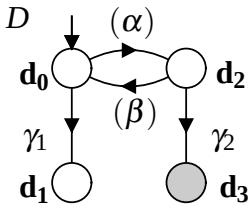


Figure 4.9: A distinguisher for the system shown in Fig. 4.8.

Example 13 *Consider again FSMs G_2, $\tilde{G}_2$, and T in Fig. 4.7. As shown in Example 12, merging states q_0 and q_2 in G_2 results in nondeterminism. To avoid this, a renaming is introduced that replaces event γ by two new events γ_1 and γ_2. Then $\tilde{G}_2$ is replaced by the abstraction G_2' in Fig. 4.8, which is a deterministic FSM. Next, since FSM T has event γ, it needs to be modified to use the new events. Thus, T is replaced by T' shown in Fig. 4.8. Now we can calculate a supervisor for G_2' and T', which is shown in Fig. 4.8 as FSM S'.*

The question that arises at this point is how the supervisor synthesised from the renamed model can control the original plant? To make this possible, a *distinguisher* is introduced in Paper 2, which is considered to be a part of the final supervisor. A distinguisher is an FSM that differentiates between the renamed events and it guarantees that only one of the renamed events is enabled at each state. Distinguishers enable the final supervisor to choose the correct transitions. For example, FSM D shown in Fig. 4.9 is a distinguisher that differentiates event γ_1 from γ_2 since it enables at most one of these events in each state. Technically, a distinguisher is the original FSM before abstraction with the renamed events to distinguish.

To see how supervisor S' and distinguisher D control the original system consisting of G_2 and T, assume that the plant wants to execute event γ in its initial state (q_0, t_0). From the renaming, we see that this event is replaced by γ_1 and γ_2. At this point, the distinguisher D is in its initial state d_0 and it only enables γ_1. However, the supervisor S' disables event γ_1 in its state s_0. Thus, the event γ is disabled at the initial state. Now assume that the plant wants to execute the event α in the state (q_0, t_0). As this event is enabled by the supervisor, the plant can execute this event and move to the state (q_2, t_0). After execution of

event α, the distinguisher goes to the state d_2 and S' stays at s_0. Now assume the plant asks if it is safe to execute event γ. Then the renaming again tells the supervisor that this event is replaced by γ_1 and γ_2. Since the distinguisher D is in the state d_2, only event γ_2 can be enabled. As the supervisor also enables the controllable event γ_2 the event γ can be executed at the global state (q_2, t_0).

In the framework of Paper 2, the supervisor is a set of FSMs. As shown above, one drawback with having supervisor as a set of FSMs is that nondeterminism should be avoided at all the steps of the compositional synthesis. Another issue with having the supervisor as a set of FSMs is that transition removal abstraction may result in a not necessarily least restrictive supervisor. These issues are resolved in the framework of Paper 3.

4.4.2 Map-Based Supervisor

As mentioned in the previous section, in the framework of Paper 2 nondeterminism is avoided at all the steps of the compositional approach. Moreover, transition removal abstractions are not used unless it becomes clear that the supervisor FSM does not need them to make a control decision, as for example selfloop-only events in Def. 17 of Paper 2. Otherwise, in the absence of a transition in the supervisor, the event of that transition, if it is controllable, will be interpreted as a disabled event. This can result in a supervisor, which is not least restrictive. Thus, transition removal abstractions are not used in Paper 2. However, there are usually much more transitions than states in a state machine, and removing transitions can significantly decrease the memory usage. These issues are resolved in Paper 3.

The following example illustrates how abstraction by removing a transition can result in not having a least restrictive supervisor FSM.

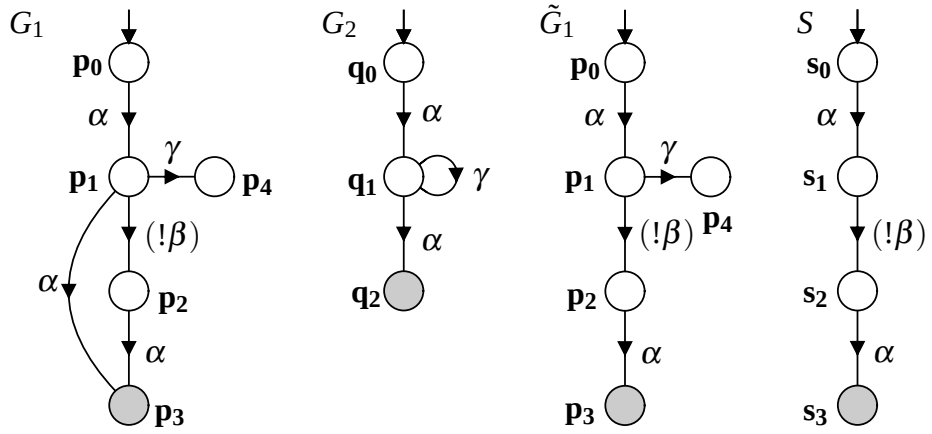
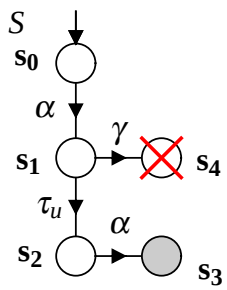


Figure 4.10: Transition removal results in the supervisor S, which is not a least restrictive supervisor.

Example 14 *Consider the system $\mathscr{G} = \{G_1, G_2\}$ in Fig. 4.10. Event α is a shared controllable event and event β is a local uncontrollable event and thus, can be hidden. The transition $p_1 \xrightarrow{\alpha} p_3$ in G_1 is a redundant transition as there is the matching path $p_1 \xrightarrow{\tau_u} p_2 \xrightarrow{\alpha} p_3$. If we remove the redundant transition the abstracted FSM $\tilde{G}_1$ is obtained as shown in Fig. 4.10. Now using FSM $\tilde{G}_1$ and G_2 to calculate a supervisor results in supervisor S shown in Fig. 4.10. This supervisor disables event α at state p_1 of G_1, while the monolithic supervisor would enable it. Thus, this supervisor is not a least restrictive supervisor.*

The problem in Example 14 happens because the supervisor is an FSM. As the supervisor works in synchrony with the plant, in the absence of the transition $s_1 \xrightarrow{\alpha} s_3$, the event α will be interpreted as disabled by the supervisor. However, as we can see from the supervisor S in Fig. 4.10, states p_1 and p_3 are considered as *safe* states. Thus, if event α happens in state p_1, this event would drive the plant to the safe state p_3. This gives us the idea to have a supervisor as a map instead of an FSM, that links the states of the original plant at the start of the compositional algorithm to the states of the current abstraction, and finally states that are determined as *safe* states by the final supervisor. This supervisor map only contains the safe states, and at each state of the plant enables controllable events, by which a state in the supervisor map is reached.

Example 15 *Consider the system $\mathscr{G} = \{\tilde{G}_1, G_2\}$ and the calculated supervisor S in Fig. 4.10. Instead of having the supervisor S as an FSM we can have the map μ shown in Fig. 4.11 as a supervisor.*



$G_1 \quad G_2$

μ

μ	(p_0,q_0)	(p_1,q_1)	(p_2,q_1)	(p_3,q_2)
	s_0	s_1	s_2	s_3

Figure 4.11: Supervisor map for system $\mathscr{G}$ shown in Fig. 4.10.

To show how the supervisor map controls the system, assume that the system is in the state (p_0,q_0), where the event α is enabled and would lead the system to state (p_1,q_1). To see whether this event is enabled by the supervisor or not we need to check if state (p_1,q_1) is considered safe in the supervisor map. This state corresponds to state s_1 in the map μ, which is a safe state. Thus, α should be enabled. Next, at the state (p_1,q_1) the plant enables events γ, $!\beta$ and α. Event

β is uncontrollable and must be enabled. By event γ, the system ends up in the state (p_4, q_1), which is not in the supervisor map. The absence of this state is interpreted as it being unsafe, and thus γ is disabled by the supervisor. Event α leads the system to the state (p_3, q_2). The image of this state in the map is s_3. Thus, event α is enabled by the supervisor.

The supervisor map μ shown in Fig. 4.11, in contrast to the supervisor FSM S shown in Fig. 4.10, enables event α in the state p_1 of G_1, even after removing transition $p_1 \xrightarrow{\alpha} p_3$. Thus, in the framework of Paper 3, in addition to state merging abstraction, transition removal abstraction is also used, which can decrease the memory usage.

The following section briefly explains state merging, state removal and transition removal abstractions that preserve synthesis equivalence.

4.4.3 Abstraction Methods Preserving Synthesis Equivalence

Even though it may be 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 FSMs of the system 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 [2], are not applicable for compositional synthesis. Technically, less states can be merged in the compositional synthesis compared to compositional verification, since merged states should not only have the same blocking property, they should also have the same synthesis property.

The state merging abstraction methods used in Paper 2 and 3 are based on *bisimulation* and *observation equivalence* [14]. While it is proven in [56] that bisimulation preserves synthesis equivalence, by using a counterexample it is shown in Paper 2 that observation equivalence does not. However, it is possible to strengthen observation equivalence to be applicable in compositional synthesis. For this purpose, *weak synthesis observation equivalence* is introduced in Paper 2 and also used as an abstraction method in Paper 3.

Weak synthesis observation equivalence, similarly to observation equivalence, is a state merging abstraction. In weak synthesis observation equivalence, the equivalent states should not only have the same future behaviour but also the same synthesis behaviour. This means that two states are considered equivalent if the synthesis algorithm either removes both of them or none of them. As mentioned before, states are removed either because of controllability or blocking

issues.

As can be seen in Def. 4 in Section 2.4, to consider two states observation equivalent they should have equivalent paths. If one state has a path, the other must have a *matching* path, s_1 and s_2 in the definition, containing the same shared events going to equivalent states. In weak synthesis observation equivalence, in contrast to observation equivalence, the controllability characteristics of events is important. If an uncontrollable event is going out of one of the states, then the matching path must only contain uncontrollable events. This condition makes sure that if one of the states is removed due to controllability issues, the other one will be also removed because of the controllability problem. For controllable events, more care must be taken. In the first part of the matching path, before the controllable event, states reached by controllable local events should be equivalent to the start state of the path, x_1 in Def. 4. In addition, in the path of local events after the controllable event, if a local uncontrollable transition goes out of the path, it must lead to a state equivalent to a state on the path, and shared uncontrollable transitions going out of the path must also be possible in the end state of the path and lead to an equivalent state. These conditions guarantee that if a marked state is reachable from one of the equivalent states, then from the other one a marked state can also be reached via a matching path.

The formal definition of weak synthesis observation equivalence can be found in Def. 22 of Paper 2. After applying weak synthesis observation equivalence in Paper 3, a map that links the states of the original FSM to the states of the abstracted FSM is generated.

Beside the state merging abstraction, in Paper 2 and Paper 3 *halfway synthesis* [22] and *unsupervisability removal* [57] are used, respectively. These abstraction methods identify and remove states of an FSM that synthesis will definitely remove later, no matter what the behaviour of the rest of the system is. An example of such states are blocking states in an FSM. Halfway synthesis works well in compositional synthesis. However, it does not identify all the removable states. Unsupervisability removal on the other hand removes the largest possible set of states [57]. After applying halfway synthesis, in Paper 2 the result of halfway synthesis is added to the supervisor set, and in Paper 3 a map is generated that links all the states of the original FSM to the abstracted state machine, and the map does not contain the removed states. As mentioned before, the absence of a state in the map is interpreted as the state being unsafe.

In Paper 3, in addition to all state-based abstractions, *synthesis transition removal* is used as an abstraction method. Synthesis transition removal is obtained by restricting the observation equivalence redundant transition removal, defined in Def. 5 in Section 2.4. Similar to weak synthesis observation equivalence for uncontrollable events, the local events on the matching path must all be uncontrollable. For controllable events, the local events before the controllable event

in the matching path should all be uncontrollable and any transition going out of the path after the controllable event should be τ_u and should lead to a state on the path. The formal definition of synthesis transition removal is given in Def. 18 of Paper 3.

4.4.4 Experimental Results

Both of the compositional synthesis algorithms described above have been implemented in the DES software tool Supremica [46] and applied to several large and complex industrial models taken from [47–54]. Both algorithms successfully compute supervisors, even for systems with more than 10^{17} reachable states, within a few seconds or minutes.

For most examples, the supervisor maps require less memory than the supervisor FSMs. In the few cases that a state machine-based supervisor uses less memory compared to the supervisor maps, the supervisor FSMs obtained after unsupervisability removal as well as the final supervisor have few states. However, the map-based supervisor needs to save all the maps from unsupervisability removal and weak synthesis observation equivalence. This makes the supervisor represented as a set of maps larger than the supervisor represented as a set of FSMs for these systems.

The memory usage to store a supervisor is an important aspect when it comes to implementing supervisors in memory limited devices like PLCs. As the supervisor map can be stored efficiently, it looks promising for use of supervisory control theory in industrial settings. More information about the experiments can be found in Section 7 of Paper 2 and 3.

Chapter 5

Summary of Included Papers

This chapter provides a brief summary of the papers that are included in the thesis. Full versions of the papers are included in Part II. The papers are reformatted for uniformity and increase readability.

Paper 1

> Mohajerani, Sahar; Malik, Robi; Fabian, Martin. *A Framework for Compositional Nonblocking Verification of Extended Finite-State Machines*. Invited paper to Journal of Discrete Event Dynamic Systems: Theory and Applications, special issue on WODES 2014.

This paper develops a general framework for *nonblocking verification* of discrete event systems modelled as *extended finite-state machines*. The extended finite-state machines of the system communicate both via shared events and shared variables. To alleviate the state-space explosion problem the algorithm gradually composes the components and applies conflict equivalence abstraction on the individual components and partially unfolds variables. The conflict equivalence abstraction methods used in this framework are a generalised form of the abstraction methods proposed for FSM in [2].

The algorithm has been implemented in Supremica, and has been successfully used to verify several large industrial models. It is shown to outperform two well-developed existing algorithms, both of which are described in Section 7 of the paper.

This paper subsumes Paper (m) by considering systems modelled as EFSMs that do not only communicate via shared variables but also via shared events. In addition, more experimental results are reported.

My contributions are: Developing the abstraction methods, mathematical proof of correctness, collecting experimental results, involved in implementation, authoring the paper.

Paper 2

> Mohajerani, Sahar; Malik, Robi; Fabian, Martin. *A Framework for Compositional Synthesis of Modular Nonblocking Supervisors.* IEEE Transaction on Automatic Control, 59(1):150-162, January 2014.

This paper presents a general framework for compositional *supervisor synthesis* of discrete event systems modelled as *deterministic finite-state machines* (in the paper referred to as finite-state automata). The state-space explosion is mitigated by the use of state-merging and state-removal based abstractions that preserve synthesis equivalence. The supervisor calculated in this framework is a modular, least restrictive, controllable and nonblocking supervisor. The supervisor consists of a set of FSMs that disable controllable events by operating in synchrony with each other and the plant.

The framework requires all FSMs and their abstractions to be deterministic, and thus, renaming is used to avoid nondeterminism after abstraction.

The algorithm has been implemented in Supremica and applied to compute modular supervisors for several large industrial models. It successfully computes modular supervisors, even for systems with more than 10^{14} reachable states, within 30 seconds and using no more than 640 MB of memory.

This paper subsumes Paper (d) and (e), by introducing a general framework for compositional synthesis, and adding experimental results. The proof of correctness of theorems in this paper can be found in (b).

My contributions are: Developing the abstraction methods, mathematical proof of correctness, collecting experimental results, involved in implementation, authoring the paper.

Paper 3

> Mohajerani, Sahar; Malik, Robi; Fabian, Martin. *Compositional Supervisor Synthesis with State Merging and Transition Removal.* Submitted to Automatica, 2014.

This paper proposes a framework to obtain memory-efficient *supervisors* for large discrete event systems modelled as interacting *finite-state machines* (in the paper referred to as finite-state automata). The supervisors obtained in this framework are least restrictive, controllable and nonblocking.

The approach combines state-based abstraction with transition removal abstraction to alleviate the state-space explosion problem. Moreover, hiding and nondeterminism after abstraction are supported. This becomes possible because

the supervisor in this framework is a cascaded map that represents the set of safe states.

The algorithm has been implemented in Supremica and applied to compute supervisors for several large industrial models. The performance of the algorithm is compared with the algorithm of Paper 2. For most models the supervisor map requires less memory compared to the supervisor state machine. This is important when it comes to implementing the supervisor in memory limited devices like PLCs.

This paper contains ideas from Paper (h). The proof of correctness of the theorems in the paper are given in (g).

My contributions are: Developing the supervisor map idea together with the co-authors, collecting experimental results, involved in implementation, authoring the paper.

Chapter 6

Concluding Remarks

The state-space explosion problem is the main obstacle in analysis of large discrete events systems. In brief, the problem arises when one tries to build the explicit monolithic model of the system. As the state-space of the model grows exponentially with the number of components, explicitly exploring the state-space of the system fails due to time and memory limitations. However, many systems are modular which makes it possible to use approaches that exploit the modular structure of the system. One of these approaches is the compositional approach, which uses abstraction to reduce complexity of the system before analysis.

The main contribution of this thesis is the compositional approach for verification and synthesis of discrete event systems modelled as FSMs and EFSMs.

The compositional nonblocking verification developed for FSM systems [2] is extended to consider conflict equivalence based abstractions for EFSMs communicating via both shared variables and shared events. Partial unfolding is introduced, which removes variables gradually to avoid the state-space explosion.

The compositional approach is also used for synthesising a least restrictive, controllable and nonblocking supervisor. Different kinds of abstractions that are guaranteed to preserve the final result are presented. These abstraction methods can considerably reduce the amount of states to examine, saving memory and time. The final supervisors can be presented as a set of FSMs or as cascaded state maps. The supervisor FSMs work in synchrony with the plant. In this setting however, nondeterminism and transition removal abstractions are avoided. The supervisor map on the other hand, only represents the safe states. This allows for transition removal abstraction and also nondeterminism after abstraction and generally more memory-efficient supervisors can be achieved.

All the algorithms have been implemented and applied to several large industrial models. The experiments show that all the systems can be successfully verified or synthesised by the implemented algorithms and in the case of synthesis, a memory efficient supervisor can be obtained. This is very important for the practicality of the supervisory control theory.

Chapter 7

Future Work

The work presented in this thesis develops the compositional approach for verification and synthesis of discrete event systems modelled as deterministic FSM or EFSM. There are number of directions towards future improvements and extensions.

It would be interesting to investigate the possibility of combining the compositional approach with the symbolic approach. Specifically developing abstraction methods to abstract the BDDs representing the system instead of abstracting FSMs or EFSMs. This may result in having smaller symbolic representations, allowing to treat even larger systems.

The compositional approach presented in this thesis is very general and can be applied to arbitrary systems. While this can be considered an advantage, it is possible to abstract more if the structure of the system is known. In this case, the synthesis equivalence and conflict equivalence properties do not need to consider arbitrary test state machines, it is enough to consider those tests that represent the actual rest of the system that is considered.

The compositional approach proposed here does not consider unobservable events in the models. It would be interesting to extend the approach to consider unobservability and nondeterminism.

For compositional verification, the most natural contribution is controllability verification of EFSM systems. In addition, it would be interesting to investigate the possibility of hiding events in an EFSM system even though their updates are not necessarily *true*. This allows more abstraction and consequently a more efficient algorithm.

Further development of the compositional synthesis is needed. In the compositional synthesis, in contrast to the compositional verification, only observation equivalence based abstraction are used to merge states. It would be interesting to develop abstraction methods beyond observation equivalence. Moreover, it would be interesting to generalise the present compositional synthesis algorithms to support systems modelled as interacting EFSMs.

References

[1] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.

[2] H. Flordal and R. Malik, "Compositional verification in supervisory control," *SIAM Journal of Control and Optimization*, vol. 48, no. 3, pp. 1914–1938, 2009.

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

[4] A. Giua and F. DiCesare, "Petri net structural analysis for supervisory control," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 2, pp. 185–195, Apr. 1994.

[5] J. A. Bergstra and J. W. Klop, "Process algebra for synchronous communication," *Information and Control*, vol. 60, no. 1–3, pp. 109–137, 1984.

[6] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Sep. 1999.

[7] H. Flordal, "Compositional approaches in supervisory control," Ph.D. dissertation, Chalmers University of Technology, Göteborg, Sweden, 2006.

[8] M. Sköldstam, K. Åkesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," in *Proceedings of the 46th IEEE Conference on Decision and Control, CDC '07*, Dec. 2007, pp. 3387–3392.

[9] K. T. Cheng and A. S. Krishnakumar, "Automatic functional test generation using the extended finite state machine model," in *Proceedings of the 30th ACM/IEEE Design Automation Conference*, Dallas, TX, USA, 1993, pp. 86–91.

[10] Y. Chen and F. Lin, "Modeling of discrete event systems using finite state machines with parameters," in *Proceedings of 2000 IEEE International*

Conference on Control Applications (CCA), Anchorage, Alaska, USA, 2000, pp. 941–946.

[11] J. Zhaoa, Y.-L. Chen, Z. Chen, F. Lin, C. Wang, and H. Zhang, "Modeling and control of discrete event systems using finite state machines with variables and their applications in power grids," *Systems & Control Letters*, vol. 61, no. 1, pp. 212–222, Jan. 2012.

[12] M. Teixeira, R. Malik, J. E. R. Cury, and M. H. de Queiroz, "Variable abstraction and approximations in supervisory control synthesis," in *2013 American Control Conference*, Washington, DC, USA, Jun. 2013, pp. 120–125.

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

[14] R. Milner, *Communication and concurrency*. Prentice-Hall, 1989.

[15] J.-C. Fernandez, "An implementation of an efficient algorithm for bisimulation equivalence," *Science of Computer Programming*, vol. 13, pp. 219–236, 1990.

[16] J. Eloranta, "Minimizing the number of transitions with respect to observation equivalence," *BIT*, vol. 31, no. 4, pp. 397–419, 1991.

[17] R. Kumar, V. Garg, and S. I. Marcus, "On controllability and normality of discrete event dynamical systems," *Systems & Control Letters*, vol. 17, pp. 157–168, 1991.

[18] R. Kumar and M. A. Shayman, "Centralized and decentralized supervisory control of nondeterministic systems under partial observation," *SIAM Journal of Control and Optimization*, vol. 35, no. 2, pp. 363–383, Mar. 1997.

[19] M.Heymann and F.Lin, "Discrete-event control of nondeterministic systems," *IEEE Transactions on Automatic Control*, vol. 43, no. 1, pp. 3–17, Jan. 1998.

[20] C. Zhou and R. Kumar, "A small model theorem for bisimilarity control under partial observation," in *Proceedings of American Control Conference 2005*, Portland, OR, USA, Aug. 2005, pp. 3937–3942.

[21] R. Malik, D. Streader, and S. Reeves, "Conflicts and fair testing," *International Journal of Foundations of Computer Science*, vol. 17, no. 4, pp. 797–813, 2006.

[22] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, "Compositional synthesis of maximally permissive supervisors using supervision equivalence," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 17, no. 4, pp. 475–504, 2007.

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

[24] S. Ware and R. Malik, "The use of language projection for compositional verification of discrete event systems," in *Proceedings of the 9th International Workshop on Discrete Event Systems, WODES'08*. Göteborg, Sweden: IEEE, May 2008, pp. 322–327.

[25] ——, "Compositional nonblocking verification using annotated automata," in *Proceedings of the 10th International Workshop on Discrete Event Systems, WODES'10*, Berlin, Germany, 2010, pp. 374–379.

[26] E. M. Clarke, D. E. Long, and K. L. McMillan, "Compositional model checking," in *Proceedings of the 5th IEEE Symposium on Logic in Computer Science*, 1989, pp. 353–362.

[27] A. Aziz, V. Singhal, G. M. Swamy, and R. K. Brayton, "Minimizing interacting finite state machines: A compositional approach to language containment," in *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors, ICCD '94*, 1994, pp. 255–261.

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

[29] M. H. de Queiroz and J. E. R. Cury, "Modular supervisory control of large scale discrete event systems," in *Proceedings of the 5th International Workshop on Discrete Event Systems, WODES '00*, Ghent, Belgium, Aug. 2000, pp. 103–110.

[30] K. Åkesson, H. Flordal, and M. Fabian, "Exploiting modularity for synthesis and verification of supervisors," in *Proceedings of the 15th IFAC World Congress on Automatic Control*, Barcelona, Spain, 2002.

[31] R. C. Hill, D. M. Tilbury, and S. Lafortune, "Modular supervisory control with equivalence-based abstraction and covering-based conflict resolution," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 20, no. 1, pp. 139–185, 2010.

[32] H. Zhong and W. M. Wonham, "On the consistency of hierarchical supervision in discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 35, no. 10, pp. 1125–1134, 1990.

[33] R. Song and R. J. Leduc, "Symbolic synthesis and verification of hierarchical interface-based supervisory control," in *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*. Ann Arbor, MI, USA: IEEE, Jul. 2006, pp. 419–426.

[34] R. C. Hill and D. M. Tilbury, "Incremental hierarchical construction of modular supervisors for discrete-event systems," *International Journal of Control*, vol. 81, no. 9, pp. 1364–1381, May 2008.

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

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

[37] S. B. AKERS, "Binary decision diagrams," *IEEE Transactions on Computers*, vol. 27, no. 6, pp. 509–516, 1978.

[38] R. E. Bryant, "Symbolic Boolean manipulation with ordered binary-decision diagrams," *ACM Computing Surveys*, vol. 24, no. 3, pp. 293–318, 1992.

[39] K. L. McMillan, *Symbolic Model Checking*. Boston, MA, USA: Kluwer Academic Publishers, 1993.

[40] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin, "Supervisory control of a rapid thermal multiprocessor," *IEEE Transactions on Automatic Control*, vol. 38, no. 7, pp. 1040–1059, Jul. 1993.

[41] A. Vahidi, "Efficient analysis of discrete event systems—supervisor synthesis with binary decision diagrams," Ph.D. dissertation, Chalmers University of Technology, Göteborg, Sweden, 2004.

[42] Z. Fei, "On symbolic analysis of discrete event systems modeled as automata with variables," Ph.D. dissertation, Chalmers University of Technology, Göteborg, Sweden, 2012.

[43] Z. Fei, S. Miremadi, K. Åkesson, and B. Lennartson, "Efficient symbolic supervisor synthesis for extended finite automata," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 6, pp. 2368–2375, Feb. 2014.

[44] R. Malik, "On the set of certain conflicts of a given language," in *Proceedings of the 7th International Workshop on Discrete Event Systems, WODES '04*. Reims, France: IFAC, Sep. 2004, pp. 277–282.

[45] R. Malik, D. Streader, and S. Reeves, "Fair testing revisited: A process-algebraic characterisation of conflicts," in *Proceedings of the 2nd International Symposium on Automated Technology for Verification and Analysis, ATVA 2004*, ser. LNCS, F. Wang, Ed., vol. 3299. Taipei, Taiwan: Springer, Oct.–Nov. 2004, pp. 120–134.

[46] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, "Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems," in *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES'06*. Ann Arbor, MI, USA: IEEE, Jul. 2006, pp. 384–385.

[47] J. O. Moody and P. J. Antsaklis, *Supervisory Control of Discrete Event Systems Using Petri Nets*. Kluwer Academic Publishers, 1998.

[48] B. Brandin and F. Charbonnier, "The supervisory control of the automated manufacturing system of the AIP," in *Proceedings of Rensselaer's 4th International Conference on Computer Integrated Manufacturing and Automation Technology*. Troy, NY, USA: IEEE Computer Society Press, 1994, pp. 319–324.

[49] L. Feng, K. Cai, and W. M. Wonham, "A structural approach to the non-blocking supervisory control of discrete-event systems," *International Journal of Advanced Manufacturing Technology*, vol. 41, pp. 1152–1168, 2009.

[50] R. J. Leduc, "PLC implementation of a DES supervisor for a manufacturing testbed: An implementation perspective," Master's thesis, Department of Electrical Engineering, University of Toronto, Ontario, Canada, 1996. [Online]. Available: http://www.cas.mcmaster.ca/~leduc

[51] KORSYS Project. [Online]. Available: http://www4.in.tum.de/proj/korsys/

[52] F. Lin and W. M. Wonham, "Decentralized control and coordination of discrete-event systems with partial observation," *IEEE Transactions on Automatic Control*, vol. 35, no. 12, pp. 1330–1337, Dec. 1990.

[53] S. Parsaeian, "Implementation of a framework for restart after unforeseen errors in manufacturing systems," Master's thesis, Chalmers University of Technology, Göteborg, Sweden, 2014.

[54] R. Malik and R. Mühlfeld, "A case study in verification of UML statecharts: the PROFIsafe protocol," *Journal of Universal Computer Science*, vol. 9, no. 2, pp. 138–151, Feb. 2003.

[55] G. Bouzon, M. H. de Queiroz, and J. E. R. Cury, "Exploiting distinguishing sensors in supervisory control of DES," in *Proceedings of the 7th IEEE International Conference on Control and Automation, ICCA '09*, Christchurch, New Zealand, Dec. 2009, pp. 442–447.

[56] S. Mohajerani, R. Malik, S. Ware, and M. Fabian, "Compositional synthesis of discrete event systems using synthesis abstraction," in *Proceedings of the 23rd Chinese Control and Decision Conference, CCDC 2011*, Mianyang, China, 2011, pp. 1549–1554.

[57] S. Ware, R. Malik, S. Mohajerani, and M. Fabian, "Certainly unsupervisable states," in *Proceedings of the 2nd International Workshop on Formal Techniques for Safety-Critical Systems, FTSCS 2013*, Queenstown, New Zealand, 2013, pp. 3–18.