THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Symbolic Supervisory Control of Resource Allocation Systems

ZHENNAN FEI

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2014

Symbolic Supervisory Control of
Resource Allocation Systems
ZHENNAN FEI
ISBN 978-91-7597-007-3

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 3688
ISSN 0346-718X

Department of Signals and Systems
Chalmers University of Technology
SE–412 96 Göteborg
Sweden
Telephone + 46 (0)31 – 772 1000

Typeset by the author using LaTeX.

Chalmers Reproservice
Göteborg, Sweden 2014

To my family

Abstract

Supervisory control theory (SCT) is a formal model-based methodology for verification and synthesis of supervisors for discrete event systems (DES). The main goal is to guarantee that the closed-loop system fulfills given specifications. SCT has great promise to assist engineers with the generation of reliable control functions. This is, for instance, beneficial to manufacturing systems where both products and production equipment might change frequently.

The industrial acceptance of SCT, however, has been limited for at least two reasons: (i) the analysis of DES involves an intrinsic difficulty known as the state-space explosion problem, which makes the explicit enumeration of enormous state-spaces for industrial systems intractable; (ii) the synthesized supervisor, represented as a deterministic finite automaton (FA) or an extended finite automaton (EFA), is not straightforward to implement in an industrial controller.

In this thesis, to address the aforementioned issues, we study the modeling, synthesis and supervisor representation of DES using binary decision diagrams (BDDs), a compact data structure for representing DES models symbolically. We propose different kinds of BDD-based algorithms for exploring the symbolically represented state-spaces in an effort to improve the abilities of existing supervisor synthesis approaches to handle large-scale DES and represent the obtained supervisors appropriately.

Following this spirit, we bring the efficiencies of BDD into a particular DES application domain – deadlock avoidance for resource allocation systems (RAS) – a problem that arises in many technological systems including flexible manufacturing systems and multi-threaded software. We propose a framework for the effective and computationally efficient development of the maximally permissive deadlock avoidance policy (DAP) for various RAS classes. Besides the employment of symbolic computation, special structural properties that are possessed by RAS are utilized by the symbolic algorithms to gain additional efficiencies in the computation of the sought DAP. Furthermore, to bridge the gap between the BDD-based representation of the target DAP and its actual industrial realization, we extend this work by introducing a procedure that generates a set of “guard” predicates to represent the resulting DAP.

The work presented in this thesis has been implemented in the SCT tool Supremica. Computational benchmarks have manifested the superiority of the proposed algorithms with respect to the previously published results. Hence, the work holds a strong potential for providing robust, practical and efficient solutions to a broad range of supervisory control and deadlock avoidance problems that are experienced in the considered DES application domain.

Acknowledgments

I didn't have time to write a short letter, so I wrote a long one instead.
– Samuel Langhorne Clemens (Mark Twain)

In retrospect, coming to study at Chalmers and pursuing PhD in the Automation research group are probably the best decisions I've ever made. Without them, I would never have come cross a lovely girl who later became my wife; I wouldn't have met these good friends who brought me unforgettable joy. Moreover, I wouldn't have had this opportunity to express my deep gratitude to many individuals below who helped me one way or the other in this thesis.

The foremost to mention among others, of course, is my supervisor Prof. Knut Åkesson. I am grateful to him for his guidance, caring, patience, ideas and supervision through the course of my PhD studies. Knut is not just my advisor on research, but also my mentor on my career and many aspects beyond.

I have had the greatest pleasure of being co-supervised by Prof. Spyros Reveliotis from Georgia Tech. I would like to thank him for his excellent guideline and invaluable feedback on the thesis. My deepest appreciation also goes to Prof. Bengt Lennartson for his continuous encouragement and support, and Prof. Martin Fabian for always letting his door open for all kinds of discussions about research and life.

Martin, Knut and Spyros have proofread almost every word of this thesis to make sure that all the pieces are put in the right places with the right sides up. I am very grateful for that.

All of my colleagues at the division of Automatic Control, Automation and Mechatronics deserve a word of appreciation. My special thanks go to the "DK" members in the Automation research group. They are (in alphabetic order): Amir, Daniel, Kristofer, Maziar, Mohammad, Mona, Nina, Oscar, Patrik, Petter, Sahar, Sathya. Also, I wish to express my deep appreciation to Sajed for the inspiring discussions and the sharing moments we have had while visiting Georgia Tech in 2012. Moreover, I would like to acknowledge all the administration staff, with special thanks to Lars, Madeleine, Natasha, Christine and Ingemar.

My life would have been dull if it were not the great friends I met here at Göteborg. I wish to thank Meiling, Jan-Erik, Xiangrui, Xiaolu, Songhe, Binru, Bo, Tong, Jun, Wei, Wanlu, Jingya, Gongpei, Xiaoming, Xinlin, Yutao, Yinan, Yujiao, Li, Yixiao, Xiaosong, Xiaodong, Xuezhi and many others, for all the good times we enjoyed together. I am also indebted to my best friends in China (6385.71 km away), Zhiqiang, Junlong, Xiaoping, Tong, Chao and Wei for always believing in me and backing me up there.

Finally, I would like to extend my deepest gratitude to my parents and grandma for their never-ending source of love, belief, and encouragement. Last, but certainly not least, my most generous love goes to my wife, Xuan. Words cannot express how grateful I am to you for accompanying me through these years. If I didn't have you, my life would be null. I would be a binary decision diagram without the one-terminal.

Zhennan Fei
Göteborg, May 2014

This work was carried out within the Wingquist Laboratory VINN Excellence Centre at Chalmers University of Technology. It was supported by the Vinnova/FFI project "Virtual commissioning of manufacturing systems including PLC logic".

List of Publications

This thesis is based on the following appended papers:

Paper 1 Zhennan Fei, Sajed Miremadi, Knut Åkesson, Bengt Lennartson. *Symbolic State-Space Exploration and Guard Generation in Supervisory Control Theory.* Agents and Artificial Intelligence – Communications in Computer and Information Science, by Joaquim Filipe and Ana Fred (eds), Springer, vol. 271, pp. 161–175, 2013.

Paper 2 Zhennan Fei, Sajed Miremadi, Knut Åkesson, Bengt Lennartson. *Efficient Symbolic Supervisor Synthesis for Extended Finite Automata.* IEEE Transactions on Control Systems Technology, in press, 2014.

Paper 3 Zhennan Fei, Spyros Reveliotis, Sajed Miremadi, Knut Åkesson. *A BDD-Based Approach for Designing Maximally Permissive Deadlock Avoidance Policies for Complex Resource Allocation Systems.* Submitted for a possible journal publication (under review), 2014.

Paper 4 Zhennan Fei, Spyros Reveliotis, Knut Åkesson. *Symbolic Computation of Boundary Unsafe States in Complex Resource Allocation Systems using Partitioning Techniques.* Submitted for a possible journal publication (under review), 2014.

Paper 5 Zhennan Fei, Knut Åkesson, Spyros Reveliotis. *Symbolic Computation and Representation of Deadlock Avoidance Policies for Complex Resource Allocation Systems with Application to Multithreaded Software.* Submitted to the 53rd IEEE Conference on Decision and Control (CDC), 2014.

Other publications

The following publications, authored or co-authored by the author of this thesis, are relevant but not included in the thesis:

- Martin Fabian, **Zhennan Fei**, Sajed Miremadi, Bengt Lennartson and Knut Åkesson. *Supervisory Control of Manufacturing Systems using Extended Finite Automata.* Formal Methods in Manufacturing, by J. Campos, C. Seatzu and X. Xie (eds), CRC Press / Taylor and Francis, pp. 295–314, 2014.

- Sajed Miremadi, **Zhennan Fei**, Knut Åkesson, Bengt Lennartson. *Symbolic Representation and Computation of Timed Discrete Event Systems.* IEEE Transactions on Automation Science and Engineering, vol. 11, no. 1, pp. 6–19, 2014.
- Sajed Miremadi, **Zhennan Fei**, Knut Åkesson, Bengt Lennartson. *Symbolic Supervisory Control of Timed Discrete Event Systems.* IEEE Transactions on Control Systems Technology, conditionally accepted, 2014.
- Bengt Lennartson, Francesco Basile, Sajed Miremadi, **Zhennan Fei**, Mona Noori Hosseini, Martin Fabian, Knut Åkesson. *Supervisory Control for State-Vector Transition Models–A Unified Approach.* IEEE Transactions on Automation Science and Engineering, vol. 11, no. 1, pp. 33–47, 2014.
- **Zhennan Fei**, Spyros Reveliotis, Knut Åkesson. *A Symbolic Approach for Maximally Permissive Deadlock Avoidance in Complex Resource Allocation Systems.* Proceedings of the 12th IFAC - IEEE International Workshop on Discrete Event Systems (WODES), 2014.
- **Zhennan Fei**, Knut Åkesson, Bengt Lennartson. *Modeling Sequential Resource Allocation Systems using Extended Finite Automata.* Proceedings of the 7th IEEE International Conference on Automation Science and Engineering (CASE), pp. 444–449, 2011.
- **Zhennan Fei**, Knut Åkesson, Bengt Lennartson. *Symbolic Reachability Computation using the Disjunctive Partitioning Technique in Supervisory Control Theory.* Proceedings of IEEE International Conference on Robotics and Automation (ICRA), pp. 4364–4369, 2011.
- **Zhennan Fei**, Sajed Miremadi, Knut Åkesson, Bengt Lennartson. *Efficient Symbolic Supervisory Synthesis and Guard Generation.* Proceedings of the 3rd International Conference on Agents and Artificial Intelligence (ICAART), vol. 1, pp. 106–115, 2011.
- Sajed Miremadi, **Zhennan Fei**, Knut Åkesson, Bengt Lennartson. *Symbolic Computation of Nonblocking Control Function for Timed Discrete Event Systems.* Proceedings of the 51th IEEE International Conference on Decision and Control (CDC), pp. 7352–7359, 2012.
- **Zhennan Fei**, Sajed Miremadi, Knut Åkesson, Bengt Lennartson. *A Symbolic Approach to Large-Scale Discrete Event Systems Modeled as Finite Automata with Variables.* Proceedings of the 8th IEEE International Conference on Automation Science and Engineering (CASE), pp. 502–507, 2012.
- Bengt Lennartson, Sajed Miremadi, **Zhennan Fei**, Mona Noori Hosseini, Martin Fabian, Knut Åkesson. *State-Vector Transition Model Applied to Supervisory Control.* Proceedings of the 17th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA), pp. 1–8, 2012.

List of Acronyms

BDD	–	Binary Decision Diagram
CF	–	Characteristic Function
DAP	–	Deadlock Avoidance Policy
DES	–	Discrete Event System
D/C RAS	–	Disjunctive/Conjunctive Resource Allocation System
EFA	–	Extended Finite Automata
EFSC	–	Extended Full Synchronous Composition
EST	–	Explicit State Transition
FA	–	Finite (State) Automata
FSC	–	Full Synchronous Composition
OBDD	–	Ordered Binary Decision Diagram
PLC	–	Programmable Logic Controller
PN	–	Petri Nets
RAS	–	Resource Allocation System
ROBDD	–	Reduced Ordered Binary Decision Diagram
R/W RAS	–	Resource Allocation System with Reader/Writer Locks
SCT	–	Supervisory Control Theory
TDES	–	Timed Discrete Event System

Contents

Part I

Introductory Chapters

Chapter 1

Introduction

In the aftermath of the revolution in computer technology, society nowadays is dependent on dedicated computer-aided systems more than ever to assist us in almost every aspect of daily life. From the very moment of waking up in the morning, we are surrounded with different hardware and software systems such as mobile phones, transportation facilities, communication tools, e-commerce systems, etc.

As such computer-aided devices and automation systems are widely used, designing reliable control logic is of paramount importance. While we may afford minor malfunctioning behavior occurring in our phones or personal computers, high demand on dependability and safety must be put in the design phase of certain critical systems such as air planes, industrial production systems or power plants, where failures might lead to loss of life and economic damages. A recent example of such a failure is the Ariane-5 rocket [1], that exploded on June 4, 1996, less than forty seconds after it was launched. The accident was caused by a software error in the computer that was responsible for calculating the rocket's movement. During the launch, an exception occurred when a large 64-bit floating-point number was converted to a 16-bit signed integer. The floating-point number that was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in an operand error. The same error also caused the backup computer to fail. As a result, incorrect attitude data was transmitted to the on-board computer, which caused the destruction of the rocket.

The increasing reliance on safety- and business-critical applications necessitates the development of formal methods for the rigorous modeling of the considered system and the accurate assessment of its associated properties. With respect to the modeling, classical control theory deals with systems whose behavior can be modeled using differential or difference equations. On the other hand, at a certain level of abstraction, the logical behavior of systems like automated manufacturing systems, computer networks and embedded control systems can be modeled as sequences of *events*. A system, characterized by a set of *states*, where the state evolution depends entirely on the occurrence of asynchronous events at discrete points in time, is referred to as a *discrete event system* (DES) [2], which is the scope of this thesis. A comprehensible example is the traffic light system, that can be modeled as a DES consisting of three states denoting either green, yellow or red, and a series of events indicating the alternation between the lights, e.g., the light turns green to yellow.

In the last two decades, research in formal methods has led to some promising methodologies that ease the burden of designing reliable systems. One approach towards the correctness of computer-based systems is *model checking* [3, 4], a formal verification technique allowing for properties of the considered system to be automatically verified on the basis of its model. Model checking usually terminates with a *yes* or *no* answer to the satisfiability of the model with respect to given specifications. Synthesis, on the other hand, can directly construct a desirable system where specifications are fulfilled. In 1987, Ramadge and Wonham proposed a model-based framework called *supervisory control theory* (SCT) [5, 6] for the automatic generation of reliable control logic, referred to as the *supervisor*, for DES. Having a model of the system to be controlled, the *plant*, and the model of the intended behavior of the controlled system, the *specification*, a supervisor can be automatically *synthesized* to control the plant according to the specification. In SCT, a supervisor is assumed to be *maximally permissive* in the sense that it restricts the plant behavior only when it is necessary. Furthermore, the SCT framework is facilitated by adopting graphical modeling formalisms such as *Petri nets* (PNs) [7], *finite automata* (FA) [2], *extended finite automata* (EFA) [8], that connect more explicitly the representation of the system behavior to the underlying system structure, and they are, thus, more compact and more amenable to processing during the synthesis phases.

In this thesis, besides studying and developing efficient supervisor synthesis in SCT for general DES, we also focus on a particular DES application domain – deadlock avoidance for *resource allocation systems* (RAS) [9]. In its basic positioning, this problem concerns the coordinated allocation of the system resources to a set of concurrently executing processes so that every process can eventually proceed to its completion. In particular, by utilizing the information about the current allocation of the system resources and the available knowledge about the structure of the executing process types, the applied control policy avoids the visitation of RAS states from which deadlock is inevitable. From an application standpoint, the need for deadlock avoidance arises in many contemporary systems, including material flow control of flexible automated production systems [10, 11, 12], traffic management of unmanned discrete material handling systems [13, 14, 15], traffic control of railway and urban monorail transport systems [16], and the lock allocation that takes place among the various threads of parallelized computer programs [17, 18].

The deadlock avoidance problem can be characterized in the classical SCT framework in a straightforward manner, through (i) expressing the underlying resource allocation dynamics into a deterministic finite automaton, and (ii) requesting the confinement of the RAS behavior to the subspace of this FA that is defined by its maximal strongly connected component that contains the system state where the RAS is idle and empty of any jobs. In fact, this characterization of the problem and its solution establishes also a notion of optimality for the considered problem, since the resulting policy prevents the formation of deadlock while retaining the maximum possible behavior for the underlying RAS.

1.1 Challenges

SCT has gained a lot of focus within the academic community, though its industrial acceptance has been scarce. This is a pity, since SCT has showed great promise when it comes to aiding the design and development of industrial control systems. We believe that the reasons for the scarce industrial acceptance of SCT are two-fold. One reason is the necessity to be able to treat systems of industrially interesting sizes. The other is how to efficiently represent the synthesized supervisor.

While normally a given DES application can be modeled as interacting components with each having a manageable size, the standard synthesis needs to consider the interactions of all components in order to explore possible global states, also referred to as the state-space. However, industrial systems nowadays are becoming more and more complex and each subsystem may consist of many interacting components. Synthesizing supervisors for such complex applications through the explicit state-space enumeration quickly becomes a serious impediment. As a matter of fact, the supervisor synthesis problem is NP-hard [19], and thus, it suffers from an inherent difficulty known as the *state-space explosion* problem, which makes the explicit enumeration of enormous state-spaces for industrial systems intractable due to lack of memory and time. To alleviate the state-space explosion problem, a well-known strategy is to represent the state-space of the considered DES *symbolically*. Here symbolic representation implies that the state-space is expressed by means of logic constraints or special data structures, which makes it possible to manipulate sets of states rather than single states. One such compact and efficient data structure that is employed in the thesis is the *binary decision diagram* (BDD) [20, 21], which, under the right conditions, can reach logarithmic compression of the involved state-spaces [21]. Nevertheless, the effective deployment of BDD-based symbolic computation in supervisory control remains a non-trivial task. A straightforward transformation from the explicit state-space enumeration into a BDD-based computation scheme does not result in a synthesis procedure that performs well on relevant problems. Hence, there is a need to develop more intelligent symbolic algorithms where the modularity and structure of DES are exploited more thoroughly.

In addition, even if the computation efficiency brought by the application of BDD makes it possible to synthesize the supervisor for a given industrial application, the supervisor will be symbolically represented as a BDD. Conceptually this symbolic representation of the obtained supervisor lies very far from the actual implementation in an industrial controller, such as a programmable logic controller (PLC). Since the original system models have been reformulated and encoded, it is cumbersome to relate each state with the corresponding BDD variables. Furthermore, the resultant control logic is of centralized nature, a feature that can be deemed as limiting/undesirable in the context of certain applications. These unsolved issues regarding the representation of the synthesized supervisor further prevents acceptance by industrial engineers who are typically not accustomed to DES or SCT. Therefore, it would be better if the representation of the supervisor lies closer to its actual implementation.

1.2 Objectives

Many contemporary computer-aided systems such as flexible manufacturing systems, communication systems and transportation systems can at some level of abstraction be described by events that model the transitions between different discrete states. Normally, industrial systems are complex and manual approaches are time-consuming, error prone and not practically feasible. Therefore, automated methodologies such as SCT are desired. On the other hand, to be able to design and implement consistent and reliable control functions for complex industrial applications, it is necessary to develop algorithms that have good performance and scalability.

The first objective of this thesis is to develop and enhance BDD-based symbolic algorithms for the computationally efficient development of control logic for large-scale DES to guarantee that the behaviors of systems fulfill given specifications. Having modeling the considered plant and specifications by using the provided modeling formalisms, the proposed symbolic algorithms encode the obtained model and represent its global behavior as a number of BDDs. The SCT analysis such as verification and synthesis then can be performed symbolically on these BDDs through the usage of the techniques developed in the thesis.

While the approaches mentioned above are targeted for the supervisor synthesis of general DES, this thesis also focuses on one particular DES application domain, i.e., deadlock avoidance for RAS. Briefly speaking, RAS characterize a broad class of applications where the underlying operations can be abstracted to a set of processes that contest for the engagement of the system reusable resources. In this thesis, we aim at developing a framework for the efficient synthesis of the maximally permissive DAP for RAS. Besides the employment of symbolic computation, additional efficiencies for the proposed symbolic algorithms are obtained through the exploitation of the structural properties possessed by the considered RAS classes.

The second objective of the thesis is to represent the result obtained from the proposed symbolic algorithms in a more comprehensible and transparent manner. On a basis of the previously developed procedure that generates Boolean conditions, referred to as guards, to represent the behavior of the synthesized supervisors, the thesis achieves this objective in two directions. For general DES, the guard generation procedure is tailored to work with the proposed symbolic algorithms in order to make it more applicable for industrial systems. By exploiting the structural properties of RAS, the guard generation procedure is extended to generate more comprehensible guards from the maximally permissive DAP.

1.3 Contributions

In the light of the above objectives for tackling the mentioned challenges that prevent SCT from having an industrial breakthrough, the thesis has the following contributions:

- It suggests and benchmarks several ways to partition and explore the symbolic representation of state-space under full synchronous composition of deterministic FA and/or EFA, such that the intermediate blow up of internal nodes in the BDD representation is kept small (Paper 1 and Paper 2).

- It adapts the suggested symbolic synthesis algorithms to the guard generation procedure, making them more applicable for industrially interesting applications. That is, simplified guards can be directly generated from the partitioned presentation of the state-space and attached to the original DES model (Paper 1).
- It introduces a modeling procedure for recasting the dynamics of RAS instances into the EFA modeling framework. This procedure is extended for transforming a special class of Petri nets, Gadara nets [22], that model the primitive lock acquisition and release operations of multithreaded programs, into the equivalent EFA models (Paper 3 and Paper 5).
- It proposes a series of BDD-based symbolic algorithms for the effective and computationally efficient development of the maximally permissive deadlock avoidance policy (DAP) for various RAS classes (Paper 3 and Paper 4).
- It extends the guard generation procedure by utilizing a monotonicity property possessed within the structure of RAS. By attaching them to the original RAS-modeling EFA, the generated predicates guard transitions to the RAS states that dominate some elements in the underlying state set (Paper 5).
- It implements all the proposed approaches and algorithms in Supremica [23, 24, 25], a software tool for automatic verification, synthesis and simulation of discrete event systems.

1.4 Outline

The thesis is divided into two parts. Part I serves as a general introduction to this field and puts the appended papers in context. Part II contains the appended papers that constitute the base of the thesis. In Part I, Chapter 2 presents the background material on discrete event systems in general, with focus on the supervisory control theory. Chapter 3 focuses on one discrete event system application domain, deadlock avoidance for resource allocation systems. Chapter 4 gives a overview of symbolic computation using binary decision diagrams for supervisor computation and representation. Through illustrating a number of examples, the chapter informally presents some of the symbolic algorithms that are introduced in the appended papers. A summary of contributions in the appended papers is provided in Chapter 6. Finally, some concluding remarks and future work are given in Chapter 7.

Chapter 2

Supervisory Control Theory

Discrete event systems (DES) are abstractions of real systems such as manufacturing, traffic control, material handling and embedded systems. DES behaviors are typically modeled in terms of *states* and *events*; states represent certain situations under which specific properties hold, while events represent significant occurrences that change those properties. A DES occupies a single state out of its many possible ones at each time instant, and transits to another on the occurrence of an event. Modeling DES has been facilitated by formalisms such as *finite automata* (FA) [2], *extended finite automata* (EFA) [8], *Petri nets* (PNs) [26, 2] and *process algebra* [27, 28] that connect more explicitly the representation of DES to the underlying system structure.

A main purpose of having DES models is the analysis and design of control functions to achieve some desired behavior. A *supervisor* is a control device for DES that through interaction with the controlled process dynamically restricts events from occurring, so as to keep the closed-loop system within a desired specification. In this way, the supervisor is a safety device; certain activities (events) are hindered from occurring in order to guarantee the safety of the considered system. However, at the same time as it prevents bad things from occurring, the supervisor must allow good things to occur.

Supervisory control theory (SCT) [5, 6, 29, 2], initiated by Ramadge and Wonham in the 80's, is a model-based formal framework for the automatic design of supervisors for DES, such that this supervisor interacting with the plant forms a closed-loop system that is, according to a given specification, correct by construction.

Though SCT has gained a lot of attention within the academic research communities, its industrial acceptance has been scarce. This is a pity, since SCT has a lot to offer when it comes to structuring and supporting the development of control functions for industrial applications. Probable reasons for the scarce industrial acceptance of SCT are two. One is the necessity to be able to model and analyze systems of practical sizes. The other is the representation of the synthesized supervisor.

This chapter introduces a set of preliminaries that are used in this thesis to model DES and synthesize supervisors. In particular, Section 2.1 starts with a discussion of two modeling formalisms. This is followed by Section 2.2 explaining supervisory control theory and the basic algorithms for synthesizing supervisors. With respect to the issues that prevent SCT from having an industrial breakthrough, Section 2.3 overviews a number of various approaches and techniques developed by the research communities aiming for addressing these issues. Finally, Section 2.4 closes the chapter with a short summary and some comments on the appended papers that are related to the discussions of this chapter.

2.1 Modeling Formalisms

There are a certain number of modeling formalisms that can be used to model DES. In this thesis we focus on two of them, finite automata (FA) and extended finite automata (EFA). FA are intuitive, easy to use, since states and transitions are explicitly represented. As used originally in [5], they not only conform well with SCT but also have strong theoretical and practical properties developed over years. EFA are a variant of ordinary FA where integer variables are introduced to improve the compactness of DES models. This richer structure, though with equal expressive power, enables the representation of the DES behavior in a conciser manner than the ordinary FA.

In general, DES can be non-deterministic, meaning that the occurrence of an event may lead the considered system to different states due to some internal, unobservable behavior. For DES with deterministic behaviors, on the other hand, next states are uniquely determined by the current states of the system and the events that are enabled from them. In this thesis, we are interested in deterministic systems, and hence, all models used in this work are considered to be deterministic.

2.1.1 Deterministic Finite Automata

Definition 2.1. [2] A deterministic finite automaton, denoted by A, is a six-tuple

$$A = (Q, \Sigma, \delta, \Gamma, q_0, Q^m)$$

where:

- Q is the finite set of states.
- Σ is the finite set of events, also known as the *alphabet* of A.
- $\delta\colon Q \times \Sigma \rightarrow Q$ is the *transition function*: $\delta(q, \sigma) = q'$, means that there is a transition labeled by event σ from *source* state q to *target* state q'; in general, δ is a *partial* function.
- $\Gamma\colon Q \rightarrow 2^{\Sigma}$ is the *active event function*; $\Gamma(q)$ is the set of all events σ for which $\delta(q, \sigma)$ is defined and it is called the *active event set* of A at state q.
- q_0 is the *initial* state.
- $Q^m \subseteq Q$ is the set of *marked* states.

For the sake of convenience, the word automata will be used to refer to deterministic finite automata in the sequel.

In Definition 2.1, the inclusion of Γ in A is superfluous in the sense that Γ can be derived from δ. For this reason, we will sometimes omit explicitly writing Γ when specifying an automaton if the active event set function is not central to the discussion.

For computational considerations, the transition function δ of an automaton A sometimes is better to be formulated as the *transition relation.* The elements $\langle q, \sigma, q' \rangle$ are said to be related if and only if there is a transition from q to q' by the event σ. Therefore, the transition relation of automaton A, denoted by $\mapsto$, can be defined as follows:

$$\mapsto\, = \{\langle q, \sigma, q' \rangle \in Q \times \Sigma \times Q \mid \delta(q, \sigma) = q'\}.$$

A finite sequence of events is called a *string* of events. An empty string is denoted by ε and all possible strings consisting of events from the alphabet Σ is denoted by Σ^*. Through the usage of strings of events, we can conveniently extend δ from domain $Q \times \Sigma$ to domain $Q \times \Sigma^*$ in the following recursive manner:

$$\begin{aligned}
&\delta(q, \varepsilon) = q \\
&\delta(q, s\sigma) = \delta(\delta(q, s), \sigma) \text{ for } s \in \Sigma^* \text{ and } \sigma \in \Sigma.
\end{aligned}$$

Frequently, a DES can be modeled by a set of FA. The monolithic system is then a *composition* of these subsystems. We will model the composition of two FA A_1 and A_2 by the *full synchronous composition* (FSC) [30], denoted by $A_1||A_2$. In the FSC of two FA, a common event is enabled if and only if it is enabled by each of the composed FA.

Definition 2.2. [30] Let $A_1 = (Q_1, \Sigma_1, \delta_1, \Gamma_1, q_0^1, Q_1^m)$ and $A_2 = (Q_2, \Sigma_2, \delta_2, \Gamma_2, q_0^2, Q_2^m)$ be two automata. The full synchronous composition, FSC, of A_1 and A_2 is defined as

$$A_1||A_2 = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{1||2}, \Gamma_{1||2}, (q_0^1, q_0^2), Q_1^m \times Q_2^m)$$

where $\delta_{1||2}$ is defined as

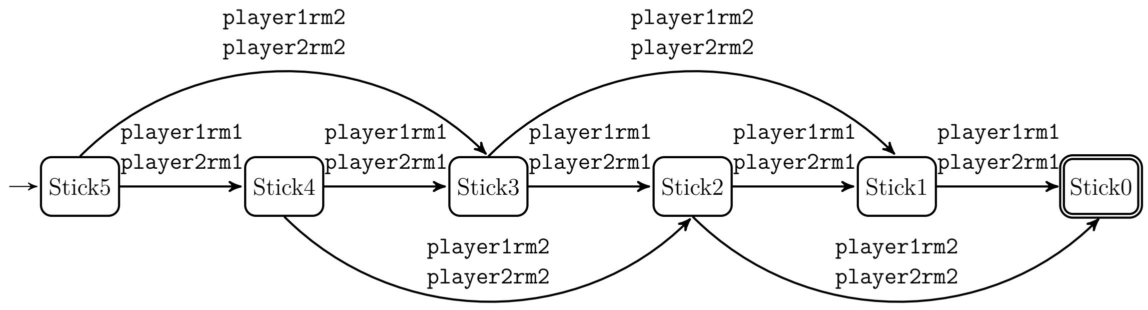
$$\delta_{1||2}((p,q),\sigma) = \begin{cases} (p', q') & \text{if } \sigma \in (\Sigma_1 \cap \Sigma_2), \delta_1(p,\sigma) = p', \delta_2(q,\sigma) = q'. \\ (p', q) & \text{if } \sigma \in (\Sigma_1 \backslash \Sigma_2), \delta_1(p,\sigma) = p'. \\ (p, q') & \text{if } \sigma \in (\Sigma_2 \backslash \Sigma_1), \delta_2(q,\sigma) = q'. \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The active event set $\Gamma_{1||2}$ follows from the definition of $\delta_{1||2}$ and is given by

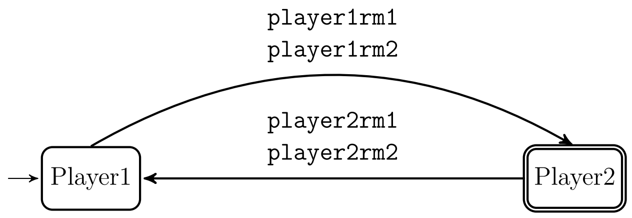
$$\Gamma_{1||2}((p,q)) = (\Gamma_1(p) \cap \Gamma_2(q)) \cup (\Gamma_1(p) \backslash \Sigma_2) \cup (\Gamma_2(q) \backslash \Sigma_1).$$

The FSC operator is both associative and commutative [2], and hence, can be extended to compose an arbitrary number of automata in a straightforward way.

We should notice from Definition 2.2 that after composition, the size of $A_1||A_2$, in terms of the number of reachable states, in the worst case is $|Q_1| \times |Q_2|$. Suppose that we compose n automata with the number of states in each automaton being



(a) Stick automaton where the state Stick0 is marked.



(b) Player automaton where the state Player2 is marked.

Figure 2.1: FA model of the stick-picking game in Example 2.1

k. The number of states of $A_1||\dots||A_n$ in the worst case is then equal to k^n. This means that the number of states grows *exponentially* as the number of components increases. We will now illustrate how FA can be used to model a simple game.

Example 2.1. DES modeling using deterministic FA is exemplified by a simple strategy game, called the stick-picking game. There are five sticks on the table, and two players take turns by removing one or two sticks from the table. The winner is the player who takes the last stick.

Figure 2.1 shows a way to model the stick-picking by deterministic FA. The depicted FA model consists of two automata that respectively capture the dynamics of the sticks and the turn-taking between two players in the game. As an example, the Stick automaton in Figure 2.1a models the way the number of sticks decreases as the game progresses, and it operates as follows. It starts in the initial state Stick5 representing the initial five sticks. Upon the occurrence of an event in $\Gamma(\text{Stick5}) = \{\texttt{player1rm1}, \texttt{player1rm2}, \texttt{player2rm1}, \texttt{player2rm2}\}$, the automaton will either transit to state Stick4 or state Stick3 depending on one or two sticks are picked from the table. This process then continues until the marked state Stick0 is finally reached. The complete (monolithic) behavior can be acquired by performing the FSC on the two automata, as shown later in Figure 2.4.

2.1.2 Extended Finite Automata

Extended finite automata (EFA) [8] are an augmentation of the ordinary FA with integer variables that are employed in a set of guard predicates and are maintained by a set of actions. A transition in an EFA is enabled if and only if its corresponding guard is true. Once a transition is taken, updating actions on the set of variables may follow. By utilizing these two mechanisms, an EFA can represent the modeled behavior in a conciser manner than the ordinary FA.

Definition 2.3. Let $v = (v_1, \ldots, v_n)$ be a set of global variables, with each variable $v_i \in v$ having the finite domain D_i. An extended finite automaton over the variable set v, denoted by E, is a five-tuple

$$E = (Q, \Sigma, \rightarrow, q_0, Q^m)$$

where:

- $Q : L \times D$ is the extended finite set of states. L is the finite set of the model *locations* and $D = D_1 \times \ldots \times D_n$ is the finite domain of the model *variables* v.
- Σ is the finite set of events, also known as the *alphabet* of E.
- $\rightarrow \subseteq L \times \Sigma \times G \times A \times L$ is the transition relation, describing a set of transitions that take place among the model locations upon the occurrence of certain events. However, these transitions are further qualified by G, which is a set of guard predicates defined on $D = D_1 \times \ldots \times D_n$, and by A, which is a collection of actions that update the variables as a consequence of an occurring transition. Each action $a \in A$ is an n-tuple of functions $(a_1, \ldots, a_n)$, with each function a_i updating the corresponding variable v_i.
- $q_0 = (\ell_0, v_0) \in L \times D$ is the initial state, where ℓ_0 is the initial location, while v_0 denotes the vector of the initial values for the variables v.
- $Q^m \subseteq L^m \times D^m \subseteq Q$ is the set of marked states. $L^m \subseteq L$ is the set of the marked locations and $D^m \subseteq D$ denotes the set of the vectors of marked values for the variables v.

In the following, we shall use the notation $\ell \overset{\sigma}{\rightarrow}_{g/a} \ell'$ as an abbreviation for $(\ell, \sigma, g, a, \ell') \in \rightarrow$. Also, the symbol ξ will be used to denote neutral actions that do not update the value of the corresponding variables; i.e., if $a_i = \xi$, action a_i does not update the variable v_i in v.

For analysis and synthesis purposes, the EFA model in Definition 2.3 is now formulated as an explicit state transition model according to the following definition.

Definition 2.4. The explicit state transition model of EFA $E = (Q, \Sigma, \rightarrow, q_0, Q^m)$ over $v = (v_1, \ldots, v_n)$, denoted by EST(E), is a five-tuple

$$\text{EST}(E) = (Q, \Sigma, \mapsto, q_0, Q^m)$$

where $\mapsto \subseteq L \times D \times \Sigma \times L \times D$ is an explicit state transition relation defined as:

$$\mapsto\, = \{(\ell, v, \sigma, \ell', v') \mid \ell \overset{\sigma}{\rightarrow}_{g/a} \ell' \in \rightarrow, v \models g, v' = a(v)\}.$$

That is, if g holds for the current values of the variables v, i.e., $v \models g$, the transition $\ell \xrightarrow{\sigma}_{g/a} \ell'$ can be taken and the values of v are consequently updated by the action a, i.e, $v' = a(v)$. Note that if variable $v_i \in v$ is not updated, i.e., $a_i = \xi$, then $v'_i = v_i$. For notational convenience, we shall let $\mapsto_{\ell \xrightarrow{\sigma}_{g/a} \ell'}$ denote the explicit representation for the transition $\ell \xrightarrow{\sigma}_{g/a} \ell' \in \rightarrow$.

An EFA E is deterministic, if whenever there exist $(\ell, v, \sigma, \ell', v') \in \mapsto_E$ and $(\ell, v, \sigma, \ell'', v'') \in \mapsto_E$, then we always have $(\ell', v') = (\ell'', v'')$. As mentioned earlier, since we merely focus on deterministic systems, in the sequel, we simply write EFA for deterministic EFA for brevity.

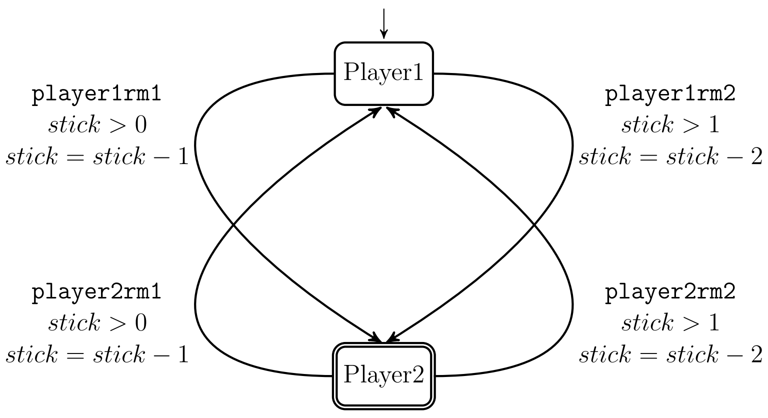


Figure 2.2: EFA model of the stick-picking game where the number of sticks is modeled as a variable *stick* with 5 as the initial value and 0 as the marked value.

Example 2.2. Figure 2.2 shows an EFA model for the stick-picking game introduced in Example 2.1. The depicted EFA contains two locations representing the respective turn of the two players in the game. Each location has two outgoing transitions labeled by events corresponding to the options of removing one or two sticks. Furthermore, an integer variable *stick* is introduced to count the number of sticks on the table. The guard predicates over the variable *stick* determine when the respective events are possible; the actions update the values of this variable when players remove sticks.

To illustrate how a transition of an EFA is represented as its explicit state transition(s), we take the upper right transition of Figure 2.2 as an example. This transition can lead to the following four explicit transitions:

$$\begin{aligned}
&(\text{Player1}, \mathit{stick} = 5, \texttt{player1rm2}, \text{Player2}, \mathit{stick} = 3),\\
&(\text{Player1}, \mathit{stick} = 4, \texttt{player1rm2}, \text{Player2}, \mathit{stick} = 2),\\
&(\text{Player1}, \mathit{stick} = 3, \texttt{player1rm2}, \text{Player2}, \mathit{stick} = 1),\\
&(\text{Player1}, \mathit{stick} = 2, \texttt{player1rm2}, \text{Player2}, \mathit{stick} = 0).
\end{aligned}$$

Analogous to the FSC for FA in Definition 2.2, the composition of two EFA is defined as the *extended full synchronous composition*. It is assumed that the variables are shared by all EFA with the same initial and marked values.

Definition 2.5. Let $E_1 = (Q_1, \Sigma_1, \rightarrow_1, q_0^1, Q_1^m)$ and $E_2 = (Q_2, \Sigma_2, \rightarrow_2, q_0^2, Q_2^m)$ be two EFA with a common variable set $v = (v_1, \dots, v_n)$. The extended full synchronous composition (EFSC) of E_1 and E_2 is defined as

$$E_1||E_2 = (Q_{1||2}, \Sigma_1 \cup \Sigma_2, \rightarrow_{1||2}, (q_0^1, q_0^2), Q_{1||2}^m)$$

where $Q_{1||2} : L_1 \times L_2 \times D$, $Q_{1||2}^m : L_1^m \times L_2^m \times D^m$, $q_0^1 = (\ell_0^1, v_0^1)$ and $q_0^2 = (\ell_0^2, v_0^2)$ with $v_0^1 = v_0^2$, and the transition $(\ell_1, \ell_2) \xrightarrow{\sigma}_{g/a} (\ell_1', \ell_2') \in \rightarrow_{1||2}$ is defined by the following three rules:

- If $\sigma \in \Sigma_1 \cap \Sigma_2$ and $\ell_1 \xrightarrow{\sigma}_{g^1/a^1} \ell_1' \in \rightarrow_{E_1}$ and $\ell_2 \xrightarrow{\sigma}_{g^2/a^2} \ell_2' \in \rightarrow_{E_2}$ are action consistent, then
  - $g = g^1 \wedge g^2$,
  - each action function a_i of a is defined as

$$a_i = \begin{cases} a_i^1 & \text{if } a_i^1 \neq \xi \text{ and } a_i^2 \neq \xi, \\ a_i^1 & \text{if } a_i^1 \neq \xi \text{ and } a_i^2 = \xi, \\ a_i^2 & \text{if } a_i^1 = \xi \text{ and } a_i^2 \neq \xi, \\ \xi & \text{if } a_i^1 = \xi \text{ and } a_i^2 = \xi. \end{cases}$$

- If $\sigma \in \Sigma_1 \backslash \Sigma_2$ and $\ell_1 \xrightarrow{\sigma}_{g^1/a^1} \ell_1' \in \rightarrow_{E_1}$ then $g = g^1$ and $a = a^1$ and $\ell_2' = \ell_2$.
- If $\sigma \in \Sigma_2 \backslash \Sigma_1$ and $\ell_2 \xrightarrow{\sigma}_{g^2/a^2} \ell_2' \in \rightarrow_{E_2}$ then $g = g^2$ and $a = a^2$ and $\ell_1' = \ell_1$.

The transitions $\ell_1 \xrightarrow{\sigma}_{g^1/a^1} \ell_1' \in \rightarrow_{E_1}$ and $\ell_2 \xrightarrow{\sigma}_{g^2/a^2} \ell_2' \in \rightarrow_{E_2}$ are *action consistent* if $a_i^1 \neq \xi$ and $a_i^2 \neq \xi$, then $a_i^1(v) = a_i^2(v)$ for all $v \in D, i = 1, \dots, n$. Note that if two transitions update any variable v_i to different values, then the composed transition is not defined. A good modeling practice is that for each variable and for each event, only one EFA is allowed to update the variable with the event, while the same variable can be updated in different EFA with any other event. In this case, the actions are structurally action consistent. Furthermore, the EFSC operator is both commutative and associative [31], and thus it can be extended to handle an arbitrary number of EFA.

2.2 Supervisory Control Theory

Given a model of the system to be controlled, the *plant*, and a *specification* of the desired controlled behavior, *supervisory control theory* (SCT) [5, 6, 29, 2] provides a formal framework to automatically compute, that is, *synthesize*, a *supervisor* that interacts with the plant to restrict the behavior such that the specification is satisfied.

Typically a plant is described by a number of sub-plants $G_1, \dots, G_n$, and the monolithic plant G is then given by $G_1|| \dots ||G_n$. Similarly, the specification K is typically described as a set of sub-specifications, $K = K_1|| \dots ||K_m$.

An important feature of the SCT is the partitioning of events as *controllable* or *uncontrollable*. The alphabet of a system thus can be partitioned into two disjoint subsets Σ^c and Σ^u, which denote the set of controllable events and the set of uncontrollable events, respectively. Note that only controllable events can be prevented from occurring by the supervisor.

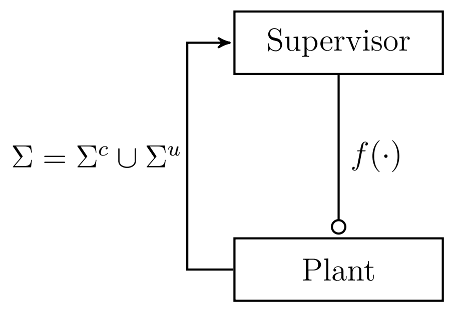


Figure 2.3: The feedback loop of the plant and the supervisor in SCT

Figure 2.3 shows the feedback loop of the plant and the supervisor. In SCT, the plant is assumed to generate all events; the supervisor, as a function $f(\cdot)$ of the observed sequence of events, can enable or disable any controllable event activated by the plant. Typically it is possible to supervise a given system in many ways, and thus, the supervisor to be synthesized can have multiple realizations depending how restrictive or permissive it is. For instance, supervisors that do not allow anything to happen might also fulfill the specification in the sense that the plant is not allowed to do anything outside the specification. However, such supervisors are not useful. Supervisors in SCT are assumed to be *maximally permissive*, which means that plants are given the greatest amount of freedom to generate events and controllable events are only disabled when necessary in order to prevent systems from reaching undesirable states where a specification is violated. The relevant literature has established conditions under which such a supervisor exists and it is unique with respect to a given plant and specification [32]. Unless otherwise noted, in this work we will only consider synthesis of supervisors with maximally permissive closed-loop behavior.

2.2.1 Supervisor Synthesis for Deterministic FA

In the Ramadge & Wonham work [5], the supervisor was realized as a control map $\Gamma : Q \to 2^{\Sigma}$. Unfortunately, the presented method for realizing such control map is inefficient. In [33] it was shown that the interaction between the plant and the supervisor can be modeled by the FSC defined in Definition 2.2. That is, the supervisor can also be considered as a deterministic FA. When a supervisor S supervises a plant G, the behavior that S will try to enforce is $G||S$. Notably, if S is not designed properly, some parts of the plant may not be amenable to the control imposed by S, so the actual behavior may be different. This is the reason why S should be synthesized using formal methods that guarantee that S never tries to control the parts of the plant that can not be controlled or, in other words, that the closed-loop behavior really is $G||S$. In this thesis, we assume that the supervisor always *refines* the plant, that is, S is structurally equivalent to $G||S$.

In addition to the basic property that the plant G under the control of the supervisor S should fulfill the given specification K, typically, there are two additional properties that supervisors are desired to have: *non-blocking* and *controllable.*

Non-blocking

Let $A = (Q, \Sigma, \delta, q_0, Q^m)$ be a deterministic finite automaton. A state $q \in Q$ is said to be coreachable to Q^m if

$$\exists s \in \Sigma^* \text{ and } q_m \in Q^m, \text{ such that } \delta(q, s) = q_m.$$

An automaton is said to be non-blocking if all of its states are coreachable to Q^m. In other words, an automaton is non-blocking if for every state there exists a path to some marked state.

Controllability

Let G and K be two deterministic finite automata and Σ^u be the set of uncontrollable events. A state $(p, q) \in Q_G \times Q_K$ is controllable if

$$\forall \sigma \in \Sigma^u : \sigma \in \Gamma_G(p) \Rightarrow \sigma \in \Gamma_{G||K}((p, q)).$$

K is controllable with respect to G if every state (p, q) in the composed system $G||K$ is controllable.

Synthesis

Given a plant G and a specification K modeled by deterministic FA, the synthesis algorithm first builds a supervisor candidate S_0 by computing the closed-loop system $G||K$. Subsequently, the algorithm removes all the undesired states from Q_{S_0} until the remaining states are both coreachable and controllable. In this thesis, the synthesis algorithm that is used to calculate the maximally permissive supervisor is called the *safe-state-synthesis*, which was introduced in [34].

Given a set of initial forbidden states Q^x that is the union of the *explicitly forbidden states* and the *initially* uncontrollable states according to the controllability explained above, the algorithm presented in Algorithm 1 executes as follows. Algorithm 1 iteratively expands the forbidden state set Q^x by adding all the states that can uncontrollably reach the existing forbidden states and/or non-coreachable states, until a final *fix-point* is reached. In particular, taking as input the state set Q^x_{i-1}, which is computed from the previous iteration, Algorithm 1 uses Algorithm 2 to compute all the states that are coreachable to the marked states in Q^m but do not belong to Q^x_{i-1}. We denote by Q' the set of these coreachable states computed at Line 4. It can be inferred that the complement of Q' with Q_{S_0}, i.e., $Q_{S_0} \backslash Q'$, contains all the states that are blocking at the current iteration. Subsequently, Algorithm 3 is utilized to identify from Q_{S_0} all the states that can reach these blocking states through uncontrollable events, and collects them into the set Q'', which later is contained in the set Q^x_i. After the fix-point computation depicted in Lines 2-7 of Algorithm 1 terminates, the safe state set Q_S can be obtained by removing the states in Q^x_i from the set Q_{S_0}. However, this state set may contain some unreachable states that need to be excluded. Hence, a forward reachability search, depicted in Algorithm 4, is employed to remove these safe states that are not reached from the initial state of S_0. The synthesis of a supervisor using the safe-state-synthesis is demonstrated in the following example.

Algorithm 1: Safe-State-Synthesis

```
  Input: $Q^x$, $Q^m$, $S_0$
  Output: The set of reachable safe states $Q_S$
1 $i := 0, Q^x_0 := Q^x$;
2 repeat
3     $i := i + 1$;
4     $Q' :=$ Restricted-Backward$(Q^m, Q^x_{i-1}, S_0)$;
5     $Q'' :=$ Uncontrollable-Backward$(Q_{S_0} \backslash Q', S_0)$;
6     $Q^x_i := Q^x_{i-1} \cup Q''$;
7 until $Q^x_i = Q^x_{i-1}$;
8 $Q_S =$ Restricted-Forward$(Q^x_i, q^{S_0}_0, S_0)$;
9 return $Q_S$;
```

Algorithm 2: Restricted-Backward

```
  Input: $Q^m$, $Q^x$ and $S_0$
  Output: The set of states that are coreachable to $Q^m$ but not in $Q^x$
1 $i := 0, Q_0 := Q^m \backslash Q^x$;
2 repeat
3     $i := i + 1$;
4     $Q_i := Q_{i-1} \cup \{q \in Q_{S_0} \mid \exists q' \in Q_{i-1}, \exists \sigma \in \Sigma_{S_0}$ s.t. $\delta_{S_0}(q, \sigma) = q'\} \backslash Q^x$;
5 until $Q_i = Q_{i-1}$;
6 return $Q_i$;
```

Algorithm 3: Uncontrollable-Backward

```
  Input: $Q_x$ and $S_0$
  Output: The set of expanded uncontrollable states
1 $i := 0, Q^x_0 := Q^x$;
2 repeat
3     $i := i + 1$;
4     $Q^x_i := Q^x_{i-1} \cup \{q \in Q_{S_0} \mid \exists q' \in Q^x_{i-1}, \exists \sigma_u \in \Sigma^u_{S_0}$ s.t. $\delta_{S_0}(q, \sigma_u) = q'\}$;
5 until $Q^x_i = Q^x_{i-1}$;
6 return $Q^x_i$;
```

Algorithm 4: Restricted-Forward

```
Input: $Q_x$, $q_0$ and $S_0$
Output: The set of states that are reachable from $q_0$
1 $i := 0, Q_0 := \{q_0\}$;
2 repeat
3     $i := i + 1$;
4     $Q_i := Q_{i-1} \cup \{q' \in Q_{S_0} \mid \exists q \in Q_{i-1}, \exists \sigma \in \Sigma_{S_0}$ s.t. $\delta_{S_0}(q, \sigma) = q'\} \backslash Q_x$;
5 until $Q_i = Q_{i-1}$;
6 return $Q_i$;
```

Example 2.3. Let us consider again the FA model of the stick-picking game introduced in Example 2.1. The objective is to design a strategy for controlling how Player1 picks sticks, in order to win the game. On the other hand, the number of sticks taken by Player2 cannot be controlled. Therefore, all events associated with Player2 are modeled as uncontrollable events, and thus, they cannot be disabled by the supervisor.

As stated earlier, a first candidate of the supervisor is the composed automaton S_0 = Player||Stick, shown in Figure 2.4. Initially, we notice that no forbidden state is included in the set Q^x, i.e., $Q^x = \emptyset$, because (i) there are no explicitly forbidden states, and (ii) no specification is involved in the considered FA model. We then apply Algorithm 1 to the automaton Player||Stick. In the first iteration, the sets Q', Q'' and Q^x_1 are:

$$\begin{aligned} Q' = \textsc{Restricted-Backward}(\{(\text{Player2, Stick0})\}, \emptyset) = \{&(\text{Player2, Stick0}), \\ &(\text{Player1, Stick1}), (\text{Player1, Stick2}), (\text{Player2, Stick2}), (\text{Player2, Stick3}), \\ &(\text{Player2, Stick4}), (\text{Player1, Stick5}), (\text{Player1, Stick3})\}. \end{aligned}$$

$$\begin{aligned} Q'' &= \textsc{Uncontrollable-Backward}(\{(\text{Player2, Stick1}), (\text{Player1, Stick0})\}) \\ &= \{(\text{Player2, Stick1}), (\text{Player1, Stick0}), (\text{Player2, Stick2})\}. \end{aligned}$$

$$Q^x_1 = \{(\text{Player2, Stick1}), (\text{Player1, Stick0}), (\text{Player2, Stick2})\}.$$

Since $Q^x_0 \neq Q^x_1$, a fix-point has not been reached, and the second iteration will be carried out:

$$\begin{aligned} Q' = \textsc{Restricted-Backward}(\{(\text{Player2, Stick0})\}, Q^x_1) = \{&(\text{Player2, Stick0}), \\ &(\text{Player1, Stick1}), (\text{Player1, Stick2}), (\text{Player2, Stick3}), (\text{Player2, Stick4}), \\ &(\text{Player1, Stick5})\}. \end{aligned}$$

$$\begin{aligned} Q'' = \textsc{Uncontrollable-Backward}(\{&(\text{Player2, Stick1}), (\text{Player1, Stick0}), \\ &(\text{Player2, Stick2}), (\text{Player1, Stick3})\}) = \{(\text{Player2, Stick1}), \\ &(\text{Player1, Stick0}), (\text{Player1, Stick3}), (\text{Player2, Stick2}), (\text{Player2, Stick4})\}. \end{aligned}$$

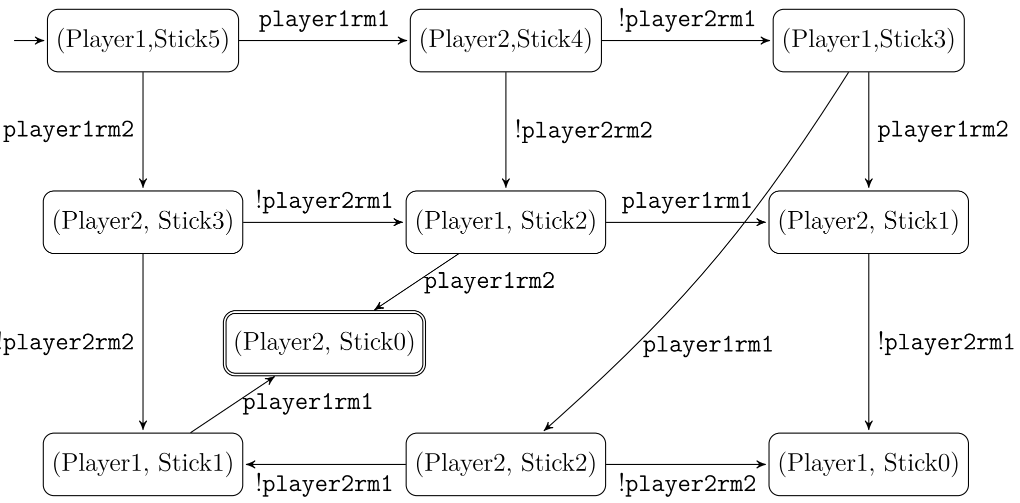


Figure 2.4: The supervisor candidate S_0 = Player||Stick where all the unreachable states are omitted for brevity. Events prefixed with exclamation marks are uncontrollable.

$$Q_2^x = \{\text{(Player2, Stick1), (Player1, Stick0), (Player2, Stick2), (Player1, Stick3),} \\ \text{(Player2, Stick4)}\}.$$

The fix-point has not been reached and thus the computations depicted in Lines 2-8 of Algorithm 1 continue as follows.

$$Q' = \textsc{Restricted-Backward}(\{\text{(Player2, Stick0)}\}, Q_2^x) = \{\text{(Player2, Stick0),} \\ \text{(Player1, Stick1), (Player1, Stick2), (Player2, Stick3), (Player1, Stick5)}\}.$$

$$\begin{aligned} Q'' &= \textsc{Uncontrollable-Backward}(\{\text{(Player2, Stick1), (Player1, Stick0),} \\ &\quad \text{(Player2, Stick2), (Player1, Stick3), (Player2, Stick4)}\}) \\ &= \{\text{(Player2, Stick1), (Player1, Stick0), (Player2, Stick2), (Player1, Stick3),} \\ &\quad \text{(Player2, Stick4)}\}. \end{aligned}$$

$$Q_3^x = \{\text{(Player2, Stick1), (Player1, Stick0), (Player1, Stick3), (Player2, Stick2),} \\ \text{(Player2, Stick4)}\}.$$

The set of forbidden states Q_3^x computed from the third iteration is the same as Q_2^x, and thus, the fix-point is reached. After removing these forbidden states and their associated transitions, the supervisor is obtained, as shown in Figure 2.5. The reader should notice that the depicted supervisor in Figure 2.5 is essentially a sub-automaton of the supervisor candidate S_0 = Player||Stick. Also, the supervisor is controllable with respect to the plant, since it never disables any of the uncontrollable events during supervision.

For a more formal and detailed explanation of the conventional supervisor synthesis, we refer readers to [6, 29, 2, 35].

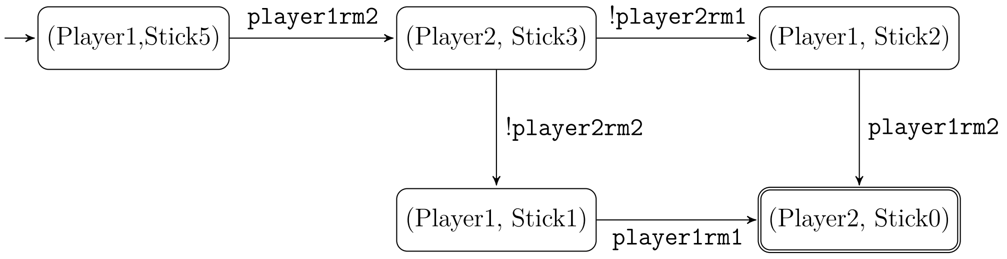


Figure 2.5: The non-blocking and controllable supervisor for the stick-picking game.

2.2.2 Transformation of EFA to FA

In the previous subsection, we explained the basic synthesis algorithm defined on DES modeled as deterministic FA, where transitions are represented explicitly by their states and events. However, due to the introduction of variables, the SCT synthesis performed on FA cannot be applied directly to EFA where states are implicitly represented. Two ways to address the issue are: (i) define a new theoretical framework for EFA, which conforms with the conventional SCT synthesis, or (ii) transform EFA to the corresponding FA and use the existing SCT framework.

In [36], the authors proposed a theoretical framework where, given a plant and a specification both modeled by EFA, the maximally permissive supervisor, represented as another EFA, can be directly synthesized.

In [8] and [37], the authors proposed an algorithm that transforms any EFA model to the corresponding FA model, such that the SCT analysis defined on FA can be reused. In particular, two kinds of automata, referred to as *location automata* and *variable automata*, are constructed by the algorithm during the transformation. The location automata preserve the same structure as the original EFA without considering the guards and actions. The variable automata, on the other hand, capture how variables are evaluated and updated for the corresponding transitions in the original EFA. More formally, given $N(= N_1 + N_2)$ EFA defined over n variables, with N_1 sub-plants $E_{G_1}, \ldots, E_{G_{N_1}}$ and N_2 sub-specifications $E_{K_1}, \ldots, E_{K_{N_2}}$, the corresponding plant and specification FA, denoted by A_G and A_K, respectively, can be computed as follows:

$$A_G = (A^{loc}_{G_1}|| \ldots ||A^{loc}_{G_{N_1}})||(A^{v_1}|| \ldots ||A^{v_n}),$$

$$A_K = (A^{loc}_{K_1}|| \ldots ||A^{loc}_{K_{N_2}})||(A^{v_1}|| \ldots ||A^{v_n}),$$

where $A^{loc}_{G_1}, \ldots, A^{loc}_{G_{N_1}}$ and $A^{loc}_{K_1}, \ldots, A^{loc}_{K_{N_2}}$ denote the transformed location FA for the EFA $E_{G_1}, \ldots, E_{G_{N_1}}$ and $E_{K_1}, \ldots, E_{K_{N_2}}$, and $A^{v_1}, \ldots, A^{v_n}$ are the transformed variable FA for the n variables.

Subsequently, based on A_G and A_K, the conventional SCT can be applied to the model. Recall that in Examples 2.1 and 2.2, we have modeled the sticking-picking game by FA and EFA, respectively. By applying the transformation algorithm in [8] and [37] to the EFA model shown in Figure 2.2, we will get the deterministic FA shown in Figure 2.4, which is actually the composition of the automata shown in Figure 2.1.

Although the algorithm introduced in [8] provides the convenience for reusing the standard SCT synthesis for DES modeled as EFA, its transformation procedure can be very time consuming, especially for complex models with many guards and actions. This is because when constructing the variable automata, the algorithm will first convert guards into the equivalent disjunctive normal forms, and then create one transition for each term of each disjunctive normal form. In the worst case, the number of transitions created by the algorithm for the variable automata can be exponential in the number of transitions included in the considered EFA. In [38] and [Paper 2], two different approaches have been introduced for directly converting an EFA model to the binary decision diagrams (BDDs) that symbolically represent the transition relation of the corresponding FA model.

2.3 Extensions of the basic SCT

This section gives an overview of different approaches and extensions of the basic supervisory control formulation.

2.3.1 Combating State-Space Explosion

The main obstacle in the analysis of DES is the *state-space explosion problem*. The problem arises when one tries to reason about the global behavior of a DES with little interaction among the components. Since standard synthesis algorithms always build the explicit state-space of the considered system $S_0 = G||K$ and store all states and transitions in the memory, they are limited by the state-space explosion problem.

The following discussion gives a number of representative approaches and techniques developed by the research community to circumvent the limitations arising from the state-space explosion problem. We notice, however, that the computational complexity of supervisor synthesis is NP-hard [19], given multiple sub-plants and sub-specifications. This means that in the worst case we cannot expect a solution that is much faster than the brute-force solution, although in many practical cases, the complexity can be alleviated by exploiting the system structure, or by employing efficient data structures to perform the synthesis procedure. The following approaches have been introduced in the literature for combating the state-space explosion problem for supervisory control problems.

Modular approaches As opposed to monolithic approaches that consider the whole system at once, modular approaches allow for the synthesis of multiple sub-supervisors, with each one controlling only parts of the plant. The advantage of modular approaches is that potentially we may never consider the full system at once, and thus the state-space explosion can be avoided. The modular synthesis was introduced in [39] and further developed and strengthened in [40, 41, 42, 43, 44, 45], among others.

Hierarchical approaches Hierarchical methods strive to divide a system into different levels of hierarchy, where the high-level supervisor is responsible for achieving a broad goal, while the low-level supervisors are responsible for the more detailed problems that are assigned by the high-level supervisor's decisions. Hierarchical supervisory control problems have been treated in [46, 47, 48, 49, 50, 51], among others.

Compositional approaches While modular and hierarchical approaches can be thought to use abstractions in different forms, compositional approaches exploit the abstraction techniques more thoroughly. As a starting point, all original components are simplified by identifying and merging certain states. Some of the abstracted components are then composed and further simplified. The process is repeated until one final relatively simple automaton is obtained. Different compositional methods developed for verification and synthesis properties in the context of SCT can be found in [52, 53, 54, 55, 56].

Symbolic representations Symbolic representations, such as binary decision diagrams (BDDs) [20, 21], represent the state-space of the underlying system symbolically. In this context, symbolic representation is used to denote that states and transitions are not represented explicitly. Instead they are represented by functions that in turn are represented using efficient data-structures. In particular, binary decision diagrams represent Boolean functions with a directed acyclic graph with a reduced number of nodes. The potential compactness offered by BDDs allows huge state-spaces to be stored within the limited amount of memory in a computer. Also, important logic operations can be performed on set of states represented using BDDs.

In the context of the SCT, BDDs were first used in [57] where the monolithic supervisor synthesis is performed symbolically on the DES model of a semiconductor manufacturing device. Later improvements have been developed in [58, 59, 60, 61, 62, 63, 64, 34, 65, 49, 66, 38] and [Paper 1, Paper 2], where the structure and modularity of systems are exploited. In [58], the authors presented an application of supervisory control theory to the control of a Rapid Thermal Multiprocessor. The plant is modeled as an input-output automaton that accepts commands as inputs, and outputs messages regarding changes that occurred in the system. A controller for the system is described in a similar way, accepting the outputs of the plant, and in turn producing commands. The controller synthesis procedure is based on BDD-based computations, which allows the automatic synthesis of realistic size manufacturing processes. In [64, 65], a hierarchical modeling formalism called state tree structure (STS) was proposed to let users model plants and specifications at different levels of detail. Taking advantage of the hierarchical structure of STS and the compactness of BDD, supervisors can be synthesized efficiently. In [66], a predicate-based synthesis and verification method for systems modeled using Hierarchical Interface-based Supervisory Control (HISC). Combined with symbolic computations using BDDs, the proposed algorithm is able to handle HISC systems with individual levels significantly larger than the methods based on explicit state enumeration. Inspired by the work presented in [67, 68], the authors of [63, 66, 34] present an approach for partitioning and exploring BDD-based representations of state-spaces for DES modeled using deterministic FA. Instead of representing the state-space of a given FA model as one monolithic BDD, the proposed approach constructs a set of BDDs that collectively represent the state-space. Based on these constructed BDDs, an efficient algorithm is proposed to explore the states iteratively to keep sizes of the relevant BDDs small. This symbolic approach later is enhanced and extended in [69] and [Paper 1], respectively. Comparing to FA, the approaches that perform the SCT synthesis procedure using BDDs directly on EFA are few. Two related works are presented in [38] and [Paper 2] where the state-space of the considered EFA can be directly represented BDDs without first transforming them into the corresponding FA. With the availability of the BDD-based representation of the state-space, the proposed algorithms can be used to synthesize the supervisor.

The approaches outlined in the previous paragraphs are not mutually exclusive, but can be combined to result in more efficient ways to compute supervisors. For example, in modular and hierarchical approaches, typically monolithic approaches are applied on subsets of components under consideration. If those subsets contain many components, compositional approaches or symbolic representations might be more applicable. Specific examples of mixing hierarchical and symbolic methods using BDDs can be found in [65, 49].

2.3.2 Supervisor as Guards

While the aforementioned approaches aim at alleviating the computational effect for designing supervisors, another issue is how to effectively represent supervisors. Typically, the synthesized supervisor for a given system is expressed as a monolithic automaton or a set of automata. For large and complex systems, such a representation of the supervisor may become incomprehensible for users. Furthermore, implementing the supervisor on a hardware platform may require more memory than available.

In [70], an approach called the *guard generation procedure* was presented to address the above issues by representing the synthesized supervisor as a set of guard predicates. The generated guards are then attached to the transitions of the original models to enforce the closed-loop behavior within the specification.

Representing supervisors as guards has several benefits. Since normally users might already have intimate knowledge of the plant, the extracted guards, despite being large in the number of terms, have a chance to be meaningful. Also, supervisors expressed as guards associated with events lie much closer to the implementation in an industrial controller, such as a programmable logic controller (PLC). Thus, the programming of the controller would be a rather simple matter of just encoding the Boolean guards in a programming language, which could even be done automatically.

Moreover, for many events the guard extraction results in no extra supervisor guards. This means that the particular event is never controlled by the supervisor, only observed, which is useful information that is not trivially obtained from a supervisor represented as automata. Similarly, some events may have supervisor guards that are always false, meaning the event is always disabled by the supervisor.

Recall that the supervisor influences the plant by dynamically disabling execution of some events from the current state, in order to avoid violation of the specification. To guarantee the correctness (i.e., controllability/non-blocking) and maximally permissiveness of the supervisor, at any state in $S_0 = G||K$, an event is either *enabled* or *disabled* from occurring. It is also possible that the execution of an event at a state of S_0 does not affect the synthesis result, e.g., if such a state is not present in the supervisor. For each event σ, we can thus generate a guard based on the states of the supervisor, indicating when σ is enabled.

Concerning the states that are retained or removed after the synthesis procedure, for each controllable event, three state sets can be considered that form the basis for generating guards [70], [Paper 1]:

1. The allowed states, where σ must be enabled in order to end up in states that belong to the supervisor.

2. The forbidden states, where σ must be disabled in order to avoid ending up in states that were removed after the synthesis procedure as the forbidden states.

3. The don't-care states, where enabling or disabling σ does not make any changes in the final supervisor.

Based on these three basic state sets, guards that indicate under which conditions the events can be executed without violating the specifications, can be extracted from the supervisor. In addition, by applying minimization methods of Boolean functions to the don't care states, and certain heuristics, the generated guards can be simplified. For a detailed explanation of the guard generation procedure and its related adoption to symbolic representations, refer to [70] and [Paper 1].

Example 2.4. As a concrete example, Figure 2.6 shows the supervisor for the stick-picking game that is represented as one additional guard attached to the EFA model in Figure 2.2. In comparison with the supervisor expressed as an automaton shown in Figure 2.5, representing the supervisor by adding this guard into the EFA model is more comprehensible and conciser.

2.3.3 Other Extensions

Partial Event Observation In the basic supervisory control theory it is assumed that all events are observable by the supervisor. This is not possible in some situations, most notably in distributed systems where each sub-supervisor is capable of observing only a subset of events generated from the shared plant.

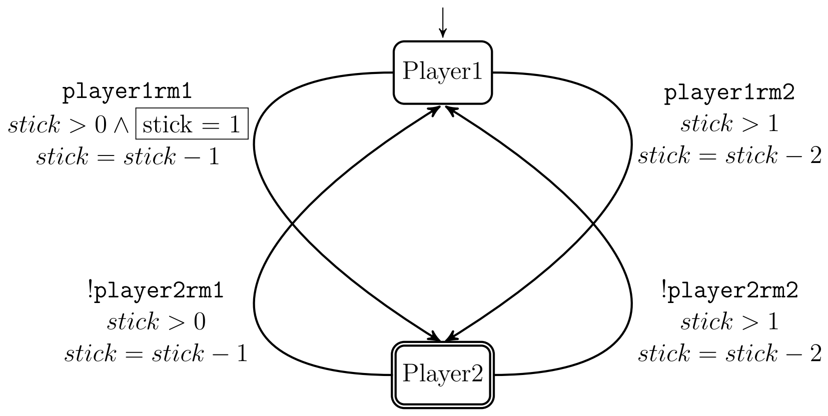


Figure 2.6: The supervisor for the stick-picking game is represented as one additional guard $stick = 1$ attached to the EFA shown in Figure 2.2. In fact, after attaching the additional guard, the whole predicate can be further reduced to $stick = 1$.

Supervisory control of distributed systems with partial observation has been treated in [71, 40, 72], among others.

Time Discrete Event Systems In DES, the logical or the *qualitative* behavior of a system is in focus. However, in some real-time applications, we also want to analyze the *quantitative* properties of a system. A DES that also considers the time instants at which the events occur, is referred to as timed DES (TDES). The supervisory control problem for TDES has been treated in [73, 74, 75, 76], and more recently in [77, 78].

2.4 Closing Remarks

This chapter presents preliminaries that are used throughout the thesis. As the modeling formalisms, FA and EFA are two major ones used to model DES. The standard SCT synthesis is originally defined and carried out on FA. An EFA is an FA extended with integer variables. This richer structure, though with equal expressive power, is more convenient for human users. However, while FA have been studies extensively in the DES research community, the approaches that perform the SCT analysis directly on EFA are few. Furthermore, the supervisor synthesis is a challenging problem due to the state-space explosion that occurs when composing a large number of individual automata.

Among the appended papers, [Paper 1] and [Paper 2] are most related to the material presented in this chapter. The former one adopts FA as the modeling formalism while the latter one uses EFA to model DES. The approaches of both papers use BDDs to represent the state-spaces of the models. Moreover, [Paper 2] aims for performing the supervisor synthesis directly on EFA while [Paper 1] focuses on how to represent supervisors that are obtained as BDDs.

Chapter 3

Deadlock Avoidance for Resource Allocation Systems

Deadlock avoidance for resource allocation system (RAS) is a particular DES application domain, which arises in many contemporary technological systems including transportation systems, workflow management, material handling and telecommunication. In its basic setting, this problem concerns the coordinated allocation of the system resources to a set of concurrently executing processes so that every process can eventually proceed to its completion. In particular, by utilizing the information about the current allocation of the system resources and the available knowledge about the structure of the executing process types, the applied control policy avoids the visitation of RAS states from which deadlock is inevitable.

Preferably, deadlock avoidance should be carried out in the *maximally permissive* manner. It is currently known [9] that the computation of the maximally permissive deadlock avoidance policy (DAP) for any given RAS can be based, in principle, on the standard synthesis procedures borrowed from the DES supervisory control theory. By expressing the underlying resource allocation dynamics as a deterministic finite automaton, we could employ the standard synthesis algorithms on this automaton to obtain the non-blocking supervisor. This supervisor would then be the maximally permissive DAP in the context of deadlock avoidance. However, as mentioned in the previous chapter, the standard SCT analysis suffers from high computational complexity. When one tries to build a complete monolithic state-space of the automaton that characterizes the actual resource allocation dynamics, state-space explosion might occur. Therefore, the realization of the optimal, or more generally, a highly permissive DAP for RAS that are encountered in contemporary applications can be a challenging task.

The rest of this chapter will evolve as follows: Section 3.1 introduces formally the concept of the resource allocation systems employed in this work. Subsequently, Section 3.2 shows how the deadlock avoidance problem is characterized and solved in the standard synthesis algorithm defined on deterministic FA. By modeling a simple RAS example as EFA, Section 3.2 also illustrates the modeling procedure that is introduced in [Paper 3]. Moreover, Section 3.2 discusses an alternative way to represent the target control policy by focusing on a set of critical states. The availability of these states enables an expedient one-step-lookahead scheme that prevents the RAS from reaching into the deadlock states. Section 3.3 overviews the state of the art methods to deal with the high computational complexity of the deadlock avoidance problem and Section 3.4 discusses some advancements or complements of this thesis to the state of the art.

3.1 Resource Allocation System

Definition 3.1. A (sequential) *resource allocation system* (RAS) is defined as a 4-tuple $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$ [9] where:

- $\mathcal{R} = \{R_1, \ldots, R_m\}$ is the set of the system *resource types* and m is the cardinality of this set.

- $C : \mathcal{R} \rightarrow \mathbb{Z}^+$ – where $\mathbb{Z}^+$ is the set of strictly positive integers – is the system *capacity* function with $C(R_i) \equiv C_i$, characterizing the number of identical units from each resource type available in the system. Resources are assumed to be *reusable*, i.e., they are engaged by the various processes according to an allocation/de-allocation cycle, and each such cycle does not affect their functional status or subsequent availability.

- $\mathcal{P} = \{J_1, \ldots, J_n\}$ denotes the set of the system *process types* supported by the considered system configuration, and n is the cardinality of this set. Each process type J_j, for $j = 1, \ldots, n$, is a composite element itself; in particular, $J_j = \langle \mathcal{S}_j, \mathcal{G}_j \rangle$, where:

  (a) $\mathcal{S}_j = \{\Xi_{j1}, \ldots, \Xi_{j,l(j)}\}$ is the set of *processing stages* involved in the definition of process type J_j and $l(j)$ denotes the number of processing stages of J_j, and

  (b) $\mathcal{G}_j$ represents some data structure communicating some sequential logic over the set of processing stages $\mathcal{S}_j$ that governs the execution of any process instance of type J_j (see Assumption 3.1 below).

- $\mathcal{A} : \bigcup_{j=1}^{n} \mathcal{S}_j \rightarrow \prod_{i=1}^{m}\{0, \ldots, C_i\}$ is the *resource allocation function*, which associates every processing stage Ξ_{jk} with the *resource allocation request* $\mathcal{A}(j,k) \equiv \mathcal{A}_{jk}$. More specifically, each $\mathcal{A}(j,k)$ is an m-dimensional vector, with its i-th component indicating the number of resource units of resource type R_i necessary to support the execution of stage Ξ_{jk}. Obviously, in a well-defined RAS, $\mathcal{A}(j,k)[i] \leq C_i, \forall j,k,i$. Also, it is assumed that $\mathcal{A}_{jk} \neq \mathbf{0}$, i.e., every processing stage requires at least one resource unit for its execution.

Moreover, the RAS class to be considered in this work satisfies the following two assumptions:

Assumption 3.1. In the considered RAS, the data structure $\mathcal{G}_j$ that defines the sequential logic of process type J_j, $j = 1, \dots, n$, corresponds to a connected *acyclic digraph* $(\mathcal{V}_j, \mathcal{E}_j)$, where the graph node set $\mathcal{V}_j$ is in one-to-one correspondence with the processing stage set $\mathcal{S}_j$. Furthermore, there are two subsets $\mathcal{V}_j^{\nearrow}$ and $\mathcal{V}_j^{\searrow}$ of $\mathcal{V}_j$ respectively defining the sets of initiating and terminating processing stages for process type J_j. The connectivity of digraph $\mathcal{G}_j$ is such that every node $v \in \mathcal{V}_j$ is accessible from the node set $\mathcal{V}_j^{\nearrow}$ and is co-accessible i.e., can reach a node to $\mathcal{V}_j^{\searrow}$. Finally, any directed path of $\mathcal{G}_j$ leading from a node of $\mathcal{V}_j^{\nearrow}$ to a node of $\mathcal{V}_j^{\searrow}$ constitutes a complete execution sequence – a "route" – for process type J_j.

Assumption 3.2. In the considered RAS, the resource allocation requests $\mathcal{A}_{jk}$, $j = 1, \dots, n$ and $k = 1, \dots, l(j)$, are "conjunctive", i.e., a processing stage Ξ_{jk} can request an arbitrary nonempty subset of the system resources for its execution. Furthermore, a process instance executing processing stage Ξ_{jk} will be able to advance to a successor processing stage $\Xi_{jk'}$, only after it has allocated the resource difference $max\{\mathbf{0}, (\mathcal{A}_{jk'} - \mathcal{A}_{jk})\}$; and it is only upon this advancement that the process will release the resource units $|min\{\mathbf{0}, (\mathcal{A}_{jk'} - \mathcal{A}_{jk})\}|$, that are not needed anymore.

For complexity considerations, we define the quantity $|\Phi| \equiv |\mathcal{R}| + |\bigcup_{j=1}^{n} \mathcal{S}_j| + \sum_{i=1}^{m} C_i$ as the "size" of RAS Φ. For notational convenience, we shall set $\mu = \sum_{j=1}^{n} |\mathcal{S}_j|$; i.e., μ denotes the number of distinct processing stages supported by the considered RAS across the entire set of its processing types. Furthermore, the various processing stages Ξ_{jk} where $j = 1, \dots, n, k = 1, \dots, l(j)$, will frequently be considered in the context of a total ordering imposed on the set $\bigcup_{j=1}^{n} \mathcal{S}_j$. In that case, the processing stages themselves and their corresponding attributes will be indexed by a single index h that runs over the set $\{1, \dots, \mu\}$ and indicates the position of the processing stages in the considered total order. Finally, given an edge $e \in \mathcal{G}_j$ linking Ξ_{jk} to $\Xi_{jk'}$, we define $e.src \equiv \Xi_{jk}$ and $e.dst \equiv \Xi_{jk'}$, i.e., $e.src$ and $e.dst$ denote respectively the source and the destination nodes of edge e.

A more general definition of the RAS concept is provided in [9]. The basic differences between Definition 3.1 and the RAS definition of [9] can be summarized as follows: First of all, the complete RAS definition involves an additional component that characterizes the time-based – or *quantitative* – dynamics of the RAS; but this component is not relevant in the modeling and analysis to be pursued in this work, and therefore, it is omitted. Moreover, the process-defining logic supported by Definition 3.1 encompasses the operational feature of routing flexibility, but it excludes the possibility of merging and splitting operations. Technically, one can classify the various RAS classes into a taxonomy that is defined on the basis of (i) the process sequencing logic of $\mathcal{G}_j$, and (ii) the resource allocation request $\mathcal{A}_{jk}$ associated with processing stages Ξ_{jk}. Then, the main RAS classes that are identified and supported by the general RAS definition of [9] are provided in Table 3.1. The reader should notice that the RAS definition in Definition 3.1 essentially corresponds to the *Disjunctive/Conjunctive* (D/C) RAS class in the taxonomy of Table 3.1.

Table 3.1: The RAS taxonomy defined in [9].

Based on the structure of the process sequencing logic	**Based on the structure of the resource allocation request**
Linear: Each process is defined by a linear sequence of stages. **Disjunctive:** A number of alternative process plans encoded by a connected digraph. **Merge-Split:** Each process is a fork/join network. **Complex:** A combination of the above behaviors.	**Single-Unit:** Each stage requires a single unit from a single resource. **Single-Type:** Each stage requires an arbitrary number of units, but all from a single resource. **Conjunctive:** An arbitrary number of units from different resources.

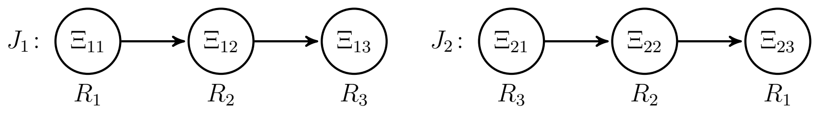


Figure 3.1: The RAS considered in Example 3.1.

Example 3.1. We demonstrate the RAS concept implied by Definition 3.1 by introducing a particular instance that will also provide an expository base for the subsequent discussion. The RAS depicted in Figure 3.1 comprises two process types J_1 and J_2, each of which is defined as a sequence of three processing stages; the stages of process types $J_j, j = 1, 2$, are denoted by $\Xi_{jk}, k = 1, 2, 3$. The system resource type set is $\mathcal{R} = \{R_1, R_2, R_3\}$, with capacity $C_i = 1$ for $i = 1, 2, 3$. The resource allocation function $\mathcal{A}$, of this RAS can be derived from the information on the process routes depicted in Figure 3.1. As a concrete example, the processing stage Ξ_{12} needs only one unit from R_2 to support its execution. Hence, $\mathcal{A}_{12}[1] = \mathcal{A}_{12}[3] = 0$, and $\mathcal{A}_{12}[2] = 1$.

3.2 Supervisory Control of RAS

The “hold-while-waiting” protocol that is described in Assumption 3.2, when combined with the arbitrary nature of the process routes and the resource allocation requests supported by the considered RAS model, can give rise to a set of RAS states. From these states, a set of processes are waiting upon each other for the release of resources that are necessary for their advancement to their next processing stage. Such persisting cyclical-waiting patterns are known as *(partial) deadlocks* in the relevant literature, and to the extent that they disrupt the smooth operation of the underlying system, they must be identified and eliminated from the system behavior. The relevant control problem is known as *deadlock avoidance*, and as remarked earlier, a natural framework for its investigation is that of SCT.

3.2.1 FA-based Modeling of RAS Dynamics

The dynamics of the RAS instance $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$, introduced in Definition 3.1, can be formally characterized by a single deterministic finite automaton (FA), denoted by $A(\Phi) = (Q, \Sigma, \delta, \Gamma, q_0, Q^m)$ that is defined as follows [9]:

1. The state set Q consists of μ-dimensional vectors $\mathbf{s}$. The components $\mathbf{s}[h], h = 1, \ldots, \mu$, of $\mathbf{s}$ are in one-to-one correspondence with the RAS processing stages, and they indicate the number of process instances executing the corresponding processing stages in the RAS state modeled by $\mathbf{s}$. Hence, Q consists of all the vectors $\mathbf{s} \in (\mathbb{Z}_0^+)^\mu$ that further satisfy

$$\forall i = 1, \ldots, m, \quad \sum_{h=1}^{\mu} \mathbf{s}[h] \cdot \mathcal{A}(\Xi_h)[i] \leq C_i \tag{3.1}$$

   where $\mathcal{A}(\Xi_h)[i] \leq C_i$ denotes the allocation request for resource R_i that is posed by the processing stage Ξ_h.

2. The event set Σ is the union of the disjoint event sets $\Sigma^{\nearrow}, \bar{\Sigma}$ and $\Sigma^{\searrow}$, where:
   (a) $\Sigma^{\nearrow} = \{e_{rp} : r = 0, \Xi_p \in \bigcup_{j=1}^{n} \mathcal{V}^{\nearrow}\}$, i.e., event e_{rp} represents the *loading* of a new process instance that starts from stage Ξ_p.
   (b) $\bar{\Sigma} = \{e_{rp} : \exists j \in 1, \ldots, n \text{ s.t. } \Xi_p \text{ is a successor of } \Xi_r \text{ in digraph } \mathcal{G}_j\}$, i.e., e_{rp} represents the *advancement* of a process instance executing stage Ξ_r to a successor stage Ξ_p.
   (c) $\Sigma^{\searrow} = \{e_{rp} : \Xi_r \in \bigcup_{j=1}^{n} \mathcal{V}^{\searrow}, p = 0\}$, i.e., e_{rp} represents the *unloading* of a finished process instance after executing its last stage Ξ_r.

3. The transition function $\delta : Q \times \Sigma \to Q$ is defined by $\mathbf{s}' = \delta(\mathbf{s}, e_{rp})$, where the components $\mathbf{s}'[h]$ of the resulting state $\mathbf{s}'$ are given by:

$$\mathbf{s}'[h] = \begin{cases} \mathbf{s}[h] - 1 & \text{if} \quad h = r \\ \mathbf{s}[h] + 1 & \text{if} \quad h = p \\ \mathbf{s}[h] & \text{otherwise} \end{cases}$$

4. The active event function $\Gamma \colon Q \to 2^\Sigma$ collects, for each state $\mathbf{s} \in Q$, the set of events $\sigma \in \Sigma$ for which the transition function $\delta(\mathbf{s}, \sigma)$ is defined (i.e., the resulting state $\mathbf{s}'$ belongs in Q).

5. The initial state q_0 is the μ-dimensional vector $\mathbf{s_0}$ set to $\mathbf{0}$.

6. The set of marked states Q^m is the singleton $\{\mathbf{s_0}\}$, where $\mathbf{s_0}$ is the initial state.

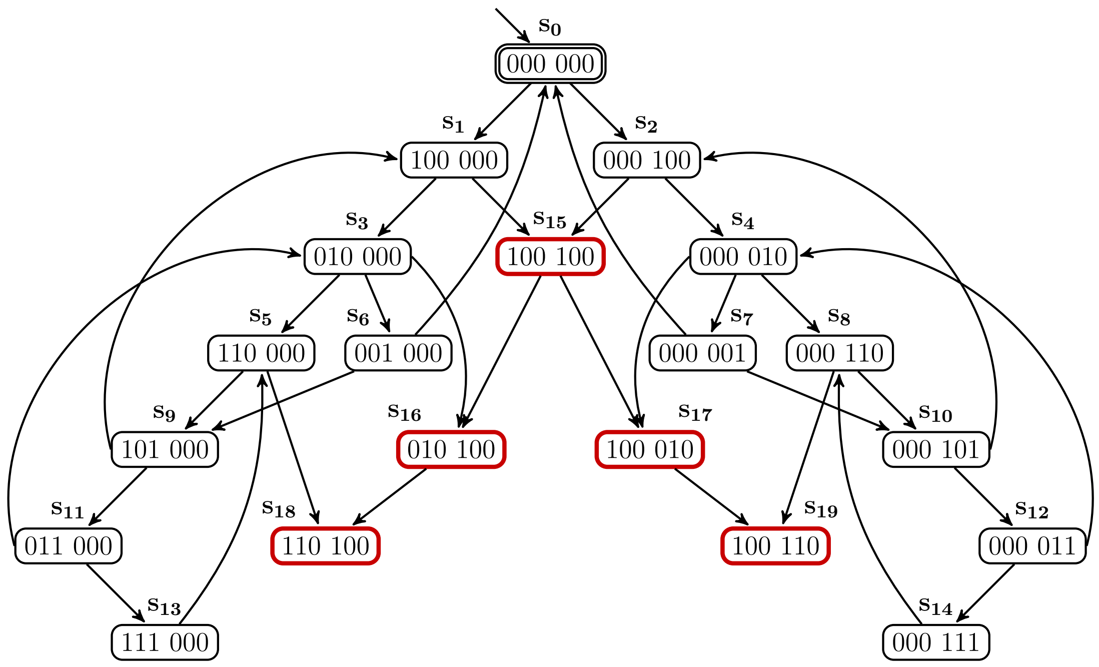
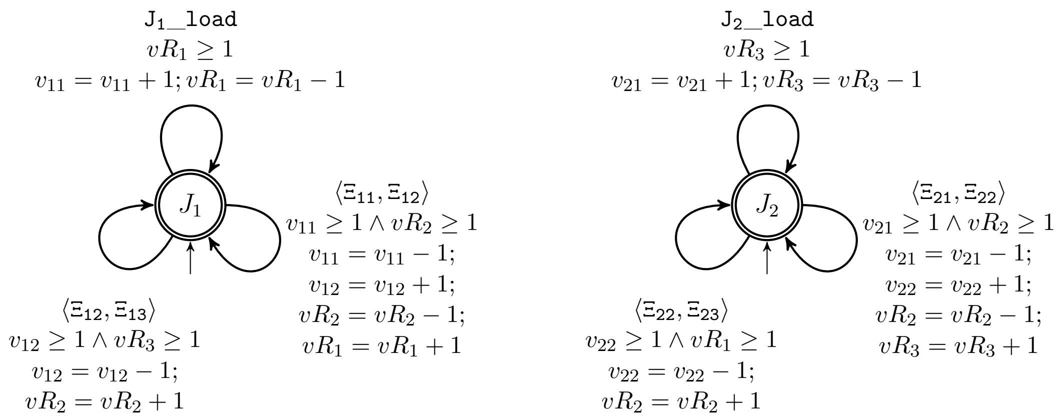


Figure 3.2: The deterministic FA modeling the RAS dynamics of Example 3.1. States depicted in red are the unsafe states.

Example 3.2. To exemplify the FA-based modeling of the RAS behavior, consider the RAS example introduced in Example 3.1. Figure 3.2 depicts the corresponding deterministic FA that models the reachable subspace of the RAS defined in Figure 3.1. The depicted FA includes the states that are reachable from the initial and marked state $\mathbf{s_0}$. The state of this RAS has six components, corresponding to each of the six processing stages; in particular, state $\mathbf{s}_0 = (0\ 0\ 0\ 0\ 0\ 0)$ denotes the initial empty state. As can be seen in Figure 3.2, the considered RAS represents 20 distinct allocation states, with the state signature ranging from 0 to 19. It will be shown shortly that the five unsafe states depicted in 3.2 are also the boundary unsafe states that define the boundary between the safe and unsafe subspaces.

3.2.2 EFA-based Modeling of RAS Dynamics

Apart from the adoption of FA, the dynamics of RAS can also be conveniently recast in the EFA modeling framework. In [Paper 3], a straightforward procedure is presented for the development of the EFA modeling the behavior of the RAS encompassed in Definition 3.1. The procedure is modular, since it models a given RAS as a set of EFA instead of a single (monolithic) automaton. These EFA when composed together using the EFSC will essentially characterize the RAS dynamics. The In this section, we illustrate this procedure by developing the EFA model for the RAS instance depicted in Figure 3.1. In the last part of the section, some additional remarks elaborate on the informational content of the generated EFA. There remarks together with the induced results will be used in Chapter 4 by the symbolic algorithms for synthesizing


J1_load
$vR_1 \geq 1$
$v_{11} = v_{11} + 1; vR_1 = vR_1 - 1$
J_1
$\langle \Xi_{11}, \Xi_{12} \rangle$
$v_{11} \geq 1 \land vR_2 \geq 1$
$v_{11} = v_{11} - 1;$
$v_{12} = v_{12} + 1;$
$vR_2 = vR_2 - 1;$
$vR_1 = vR_1 + 1$
$\langle \Xi_{12}, \Xi_{13} \rangle$
$v_{12} \geq 1 \land vR_3 \geq 1$
$v_{12} = v_{12} - 1;$
$vR_2 = vR_2 + 1$
J2_load
$vR_3 \geq 1$
$v_{21} = v_{21} + 1; vR_3 = vR_3 - 1$
J_2
$\langle \Xi_{21}, \Xi_{22} \rangle$
$v_{21} \geq 1 \land vR_2 \geq 1$
$v_{21} = v_{21} - 1;$
$v_{22} = v_{22} + 1;$
$vR_2 = vR_2 - 1;$
$vR_3 = vR_3 + 1$
$\langle \Xi_{22}, \Xi_{23} \rangle$
$v_{22} \geq 1 \land vR_1 \geq 1$
$v_{22} = v_{22} - 1;$
$vR_2 = vR_2 + 1$

(a) The EFA $E(J_1)$ modeling the process type J_1(b) The EFA $E(J_2)$ modeling the process type J_2

Figure 3.3: The EFA model for the RAS instance in Figure 3.1

the maximally permissive DAP for RAS.

Example 3.3. In the approach of [Paper 3], each process type is modeled as a distinct EFA. Figure 3.3 shows the EFA that models the behavior of the two processes J_1 and J_2 depicted in the RAS of Figure 3.1. Taking Figure 3.3a as an example, this EFA has only one location, and its three transitions correspond to the loading and the process-advancing events among the different stages. On the other hand, since a process instance that has reached its final stage can always leave the system without any further resource requests, the unloading event is modeled only implicitly through the event that models the process access to its terminal stage(s). More specifically, in the EFA depicted in Figure 3.3, the evolution of a process instance through the various processing stages is traced by the *instance variables* $v_{1j},\ j = 1, 2$; each of these variables counts the number of process instances that are executing the corresponding processing stage. The model does not include a variable v_{13} since it is assumed that a process instance reaching stage Ξ_{13} is eventually unloaded from the system, without the need for any further resource allocation action.

The two EFA $E(J_1)$ and $E(J_2)$ that model the process types J_1 and J_2 are linked through the global *resource variables* $vR_i,\ i = 1, 2, 3$, where each variable vR_i denotes the number of free units of resource R_i. The domain of variable vR_i is $\{0, \dots, C_i\}$.

Since, under proper RAS operation, the initial and the final state correspond to the empty state, both the initial and the marked values of each variable vR_i are equal to C_i, and the corresponding values for all instance variables v_{jk} are equal to zero.

Finally, as depicted in Figure 3.3, the resource and the instance variables are used to construct the necessary guards and actions for the system transitions. The guards determine whether a process-loading or advancing event can take place, on the basis of the process and the resource availability. Upon the occurrence of such an event, the corresponding actions update accordingly the available resource units and the process instances that are active at the various processing stages. For a more detailed discussion on the EFA-based modeling of RAS and the above example, the reader is referred to [Paper 3].

We remind the reader that every legitimate resource allocation state of the considered RAS must adhere to the restrictions that are imposed by the limited capacities of the system resources. In the representation of the EFA modeling an RAS instance Φ, these restrictions are expressed by the following constraints

$$\forall i \in \{1, \ldots, m\}, \quad vR_i + \sum_{j=1}^{n} \sum_{k=1}^{l(j)-1} \mathcal{A}_{jk}[i] * v_{jk} = C_i \tag{3.2}$$

In (3.2), we have taken into consideration the fact that terminal processing stages are not explicitly accounted for in the considered EFA model (for the reasons explained earlier). From a more technical standpoint, the constraints of (3.2) can be perceived as a set of resource-induced *invariants* that must be observed by the dynamics of the EFA model in order to provide a faithful representation of the actual RAS dynamics. Hence, in the following, we shall characterize a state $\mathbf{s}$ in the RAS dynamics modeled by the EFA model with a variable vector v satisfying the constraints of (3.2) as a *feasible* state.

Figure 3.4 shows the explicit state transition (EST) model of $E(J_1)||E(J_2)$, which explicitly describes the actual dynamics of the RAS instance Φ irrespective of the resource allocation and deallocation at the last processing stages. As can be seen in Figure 3.4, the depicted EST model involves twelve (12) states, with each state $\mathbf{s_i}$, $i = 0, \ldots, 11$, being described by seven components that correspond to the values of the instance variables $v_{11}, v_{12}, v_{21}, v_{22}$ and the resource variables vR_1, vR_2, vR_3. Notice that the depicted EST model is different from the deterministic FA shown in Figure 3.2, where the states do not have components representing the system resources, and the resource allocation and deallocation behaviors for the last processing stages are explicitly modeled.

3.2.3 The Target Maximally Permissive DAP

By expressing the dynamics of a RAS instance Φ by the corresponding deterministic FA, $A(\Phi)$, we have translated the problem of designing the maximally permissive DAP for Φ into a supervisory control problem, and thus, the synthesis algorithms such as the safe-state-synthesis algorithm can be applied to synthesize the supervisor. Since neither uncontrollable events nor any initial forbidden states are defined, the resulting supervisor for $A(\Phi)$ can be simply obtained by performing the non-blocking synthesis on this automaton with respect to its empty state $\mathbf{s_0}$, that defines the initial and the target state for any successful operational cycle of the underlying RAS.

In the relevant RAS theory, states that are coreachable to the RAS idle and empty state $\mathbf{s_0}$ are also characterized as *safe*, and, correspondingly, states that are not coreachable are characterized as *unsafe*.

The RAS unsafety characterized in the previous paragraph results from the formation of (partial) *deadlocks* among a (sub-)set of the running processes. The unsafe

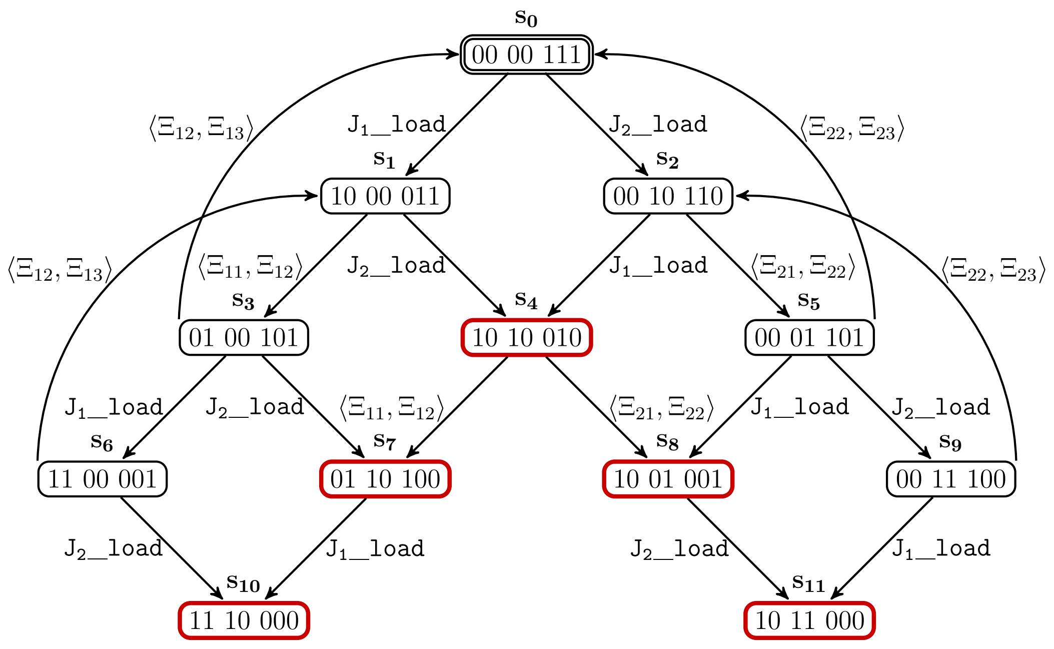


Figure 3.4: The EST model of the composition of two EFA depicted in Figure 3.3. The depicted EST model includes only the RAS feasible states that are reachable from the initial and target state $\mathbf{s}_0$. States depicted in red are unsafe.

states are essentially those states from which the formation of deadlock is unavoidable. A formal definition of this concept is as follows:

Definition 3.2. A (partial) deadlock is a RAS state where a (sub-)set of processes are entangled in a circular waiting pattern with each process in this set requiring, in order to advance to its next processing stage, some resource units that are held by other processes in the set.

For notational convenience, we denote the sets of safe states, unsafe states and deadlock states by S, U and D. Clearly, $S \cap U = \emptyset$; $D \subseteq U$.

Example 3.4. In the deterministic FA $A(\Phi)$ depicted in Figure 3.2, the states $\mathbf{s_{15}}$ – $\mathbf{s_{19}}$ are identified as the unsafe states; the remaining states are safe. The non-blocking supervisor of this automaton can be obtained through the removal of the unsafe states. Also, among these five unsafe states, the deadlock states are $\mathbf{s_{16}}$ – $\mathbf{s_{19}}$. On the other hand, the state $\mathbf{s_{15}}$ does not contain deadlock, since two processing-advancing events are enabled from the state.

When it comes to the implementation of the control function, we could have employed the FA representation of the controlled system behavior under the target DAP. However, for many practical RAS configurations, the explicit storage and on-line parsing of this information is of prohibitive computational cost due to the enormous number of the involved states.

Alternatively, the maximally permissive DAP can be implemented by a one-step–lookahead scheme that prevents the RAS from reaching outside its safe subspace. In particular, a subset of unsafe states, referred to as *the set of boundary unsafe states*, can be employed for the partition of the safe and unsafe subspaces. The set of boundary unsafe states, B, can be expressed as $B \equiv \{\mathbf{s}' \in U \mid \exists \mathbf{s} \in S, \exists \sigma \in \Sigma \text{ s.t. } \delta(\mathbf{s}, \sigma) = \mathbf{s}'\}$. Furthermore, the entire set of boundary unsafe states can be represented by its minimal elements [79], denoted by the set $\overline{B}$, since the notion of unsafety presents a monotonicity property that endows this set with properties similar to those of a right-closed set [80]. Consequently, the availability of the set of minimal boundary unsafe states enables an expedient one-step-lookahead scheme preventing the RAS from reaching outside its safe region. In particular, any tentative transitions taking the underlying RAS to a state that dominates, component-wise, some minimal boundary unsafe state will be disabled by the target DAP.

Example 3.5. As a concrete example, the boundary unsafe state set B and the minimal boundary unsafe state set $\overline{B}$ for the deterministic FA depicted in Figure 3.2 are $B = \{\mathbf{s_{15}}, \mathbf{s_{16}}, \mathbf{s_{17}}, \mathbf{s_{18}}, \mathbf{s_{19}}\}$, and $\overline{B} = \{\mathbf{s_{15}}, \mathbf{s_{16}}, \mathbf{s_{17}}\}$.

In [Paper 5], inspired by the guard generation procedure introduced in [70], to represent the sought DAP, we extend the symbolic framework that is proposed in [Paper 3] and [Paper 4] by introducing a straightforward procedure that generates a set of comprehensible guards from the set of minimal boundary unsafe states. By attaching these guards to the original model, the generated predicates guard transitions to the states that dominate some elements in the set of minimal boundary unsafe states. For a detailed exposition of this procedure and its application to the establishment of deadlock avoidance of a multithreaded program, refer to [Paper 5].

3.3 Dealing with the NP-Hardness of the Maximally Permissive DAP

A significant body of results that are currently available in the relevant literature concerns the computational complexity of the deadlock avoidance problem and the computation of the maximally permissive DAP. Along these lines, it has been established that computing the maximally permissive DAP is NP-hard for the majority of RAS behavior [81, 82, 83]. In this section we overview the main approaches that have been employed by the research community in its effort to cope with this non-polynomial complexity of the maximally permissive DAP.

The research community has tried to circumvent the limitations arising from these negative results by pursuing the following directions:

1. The identification of "special structure" that allows the deployment of the maximally permissive DAP through algorithms of polynomial complexity with respect to the size $|\Phi|$ of Definition 3.1. Some representative results can be found in [81, 84, 85, 12, 83].

2. The development of sub-optimal – i.e., non-maximally permissive – solutions that are based on state properties that are polynomially decidable with respect to the RAS size $|\Phi|$. Under these policies, a tentative RAS transition is allowed only if the resultant state satisfies the policy-defining property, which acts as a surrogate characterization of safety. Hence, if the reachable subspace satisfying such a policy-defining property constitutes a strongly connected component containing $\mathbf{s}_0$ in the deterministic FA that models the RAS dynamics, the system behavior will remain deadlock free. Specific examples implementing this general idea can be found in [86, 87, 11, 88, 89, 90].

3. The adoption of alternative, compact representations of the considered RAS dynamics in the hope that the compactness of these alternative representations, combined with further structural properties and insights revealed by them, will also lead, at least in most practical cases, to fairly compact characterizations of the target policy and to more efficient approaches for its derivation. A modeling framework that seems to hold particular promise along this line of research, and therefore, has been explored more persistently in the past, is that of Petri nets (PNs) [7]. Since the RAS structural analysis and the design of DAPs by means of PN modeling framework is beyond the scope of the thesis, we forego any further discussion in this research and refer readers to some indicative works presented in [10, 91, 92, 93, 94]. We notice, however, that the PN-based approaches for the realization of the maximally permissive DAP essentially seek to express this DAP as a set of linear inequalities to be imposed upon the RAS state. As it is established in the relevant literature [95, 9, 96], such a representation will not be always possible in the considered RAS class. Hence, these approaches are inherently limited in their ability to effectively represent and compute the maximally permissive DAP.

4. Another prominent approach that has been developed primarily in the context of PN modeling, but essentially spans, both, the FA and PN-based representations, is that of the "theory of regions" [97] and its derivatives. The key problem addressed by the theory of regions is the conversion of a system modeled originally as an FA into a Petri net such that each distinct event is represented by a single transition, and the reachability graph of the synchronized Petri net is isomorphic to the original FA. In [98], it was proposed to use the theory of regions to design a PN-based representation of the maximally permissive DAP by first computing the maximally permissive DAP using enumerative FA-based approaches borrowed from SCT; the obtained policy is subsequently encoded to a more compact PN model through the theory of regions. The approaches in [98, 99] can find the optimal supervisor if such a supervisor exists. But these approaches are also limited by the aforementioned potential inability to express the maximally permissive DAP as a PN. Furthermore, even in their feasible cases, practical experience has shown that these methods are very demanding from a computational standpoint, and they result in PN-based representations

of the maximally permissive DAP that are much larger than the PN modeling the original RAS.

5. The above discussion has revealed a number of limitations regarding the deployment of the maximally permissive DAP by using the theory of regions. In order to address these limitations, an alternative line research has been proposed, leading to effective and efficient implementations of the maximally permissive DAP for pretty sizable RAS instances. Similar to the approach of the theory of regions, the proposed approaches in this line of research are decomposed to two stages, with the first stage obtaining this policy in the SCT framework and the second stage trying to express the obtained result in a more compact form. However, instead of explicitly relying on the results from the PN-based modeling framework, the optimal DAP is perceived as a classifier that dichotomizes the RAS state-space into its safe and unsafe subspaces. The methods that are pursued in this line of research open new ways for thinking about the considered problem, that complement effectively all the previously used approaches. To develop computationally efficient classifications for the RAS classes under consideration, *parametric* or *non-parametric* representations can be selected for the sought classifier. Roughly speaking, a parametric classifier is defined by a set of linear inequalities and/or Boolean functions, whereas a non-parametric classifier is defined by a pertinent data structure that stores the information needed for the classification. For parametric classifiers, the classifier-design problem is defined as a minimization problem over a certain parameter space that results from the adopted representation. Specific approaches for designing parametric classifiers for various RAS classes are presented in [100], [101], [80], [102], [103] and [104].

6. A more recent approach that is introduced in [79] develops and deploys the non-parametric representation for the sought classifier through the identification and the efficient storage of the *minimal boundary unsafe states* of the underlying RAS state-space. As already noted earlier, these critical states define the boundary between the safe and unsafe subspaces, with a tentative transition considered to be unsafe if the resulting state is greater than or equal, componentwise, to one of the minimal boundary unsafe states. Furthermore, the results of [105] complement the work of [79] by introducing an algorithm that enumerates all the minimal unsafe states while avoiding the complete enumeration of the RAS state-space. The key idea for the algorithm of [105] stems from the remark that, in the considered RAS dynamics, unsafety is defined by unavoidable absorption into the system deadlocks. Hence, the unsafe states of interest can be retrieved by a localized computation that starts from the RAS deadlocks and backtraces the RAS dynamics until it hits the boundary between the safe and unsafe subspaces. Numerical experimentation has established the ability of this promising approach to effectively compute the maximally permissive DAP for RAS with large structure and state-spaces.

More extensive and comprehensive treatments of many of the results and methodologies that were outlined in the previous paragraph, can be found in [95, 9, 106, 96, 104, 107].

3.4 Closing Remarks

As one particular DES application domain, the deadlock avoidance problem can be characterized in the standard SCT framework in a straightforward manner. First, the underlying resource allocation dynamics is expressed into a deterministic FA. Then the standard non-blocking supervisor synthesis is carried out on this FA to obtain the maximally permissive DAP. To deploy the maximally permissive DAP for RAS efficiently, the notion of classifier is introduced and defined in the literature. Both parametric and non-parametric representations of the sought classifiers are developed. A parametric classifier is defined by a set of linear inequalities and/or Boolean functions, whereas a non-parametric classifier is defined by a pertinent data structure that stores the information needed for the classification.

To put the work of this thesis in context, the framework presented in [Paper 3] and [Paper 4] introduces two different ways to develop non-parametric classifiers for RAS, by symbolically representing the RAS dynamics as BDDs and computing the minimal boundary unsafe states. [Paper 5] extends the framework by introducing a procedure that generates a set of guards from the BDD representing all the minimal boundary unsafe states for a given RAS. This set of generated guards can be perceived as a comprehensible representation of the non-parametric classifier.

Chapter 4

Symbolic Representation and Computation

Symbolic representation and computation have been believed as an effective way to attack the state-space explosion problem. Here "symbolic" implies that the state-space of a given DES is represented by means of logic constraints or special data structures. Accordingly, the supervisor synthesis procedure can be reformulated in a way that sets of states and sets of transitions are manipulated rather than single states and transitions.

There are several possibilities to realize the standard synthesis algorithm in a set-based setting. The most prominent ones rely on a binary encoding of the states, which permits identifying subsets of the state-space and the transition relation with *characteristic functions*. To obtain compact representations of characteristic functions, special data structures have been developed, and among them, the most well-known one is the *binary decision diagram* (BDD) [20, 21]. Representing the state-space of DES through BDDs has several benefits. Apart from the compact representations, BDDs are easy to manipulate; all usual logical operations such AND, OR, NOT, NAND, etc. can be performed directly on BDDs. Also, a BDD with a fixed variable ordering is a canonical representation of a Boolean function, which makes it easy to check equivalence for Boolean functions.

However, symbolic computation using BDDs is not a silver bullet. First, BDDs are sensitive to variable orderings, and finding the best variable ordering is an NP-complete problem [108]. Furthermore, a straightforward transformation from the explicit state-space enumeration into a BDD-based computation scheme does not guarantee that the synthesis procedure will become remarkably efficient. Hence, there is a need to develop more intelligent symbolic algorithms where the modularity and structure of DES are exploited more thoroughly.

In this chapter, first some general discussions about BDDs is given in Section 4.1. This is followed by Section 4.2 that describes some contributions of this thesis to the symbolic representations of the state-spaces of DES modeled as FA and EFA. Section 4.3 focuses on the state-space exploration, the most fundamental challenge for supervisor synthesis. Moreover, the section briefly discusses and illustrated a number of new techniques for computing the supervisors of DES and the maximal permissive DAPs for RAS. Finally, Section 4.4 discusses the experimental results that demonstrate the extensive computational gains obtained by the proposed algorithms.

4.1 Binary Decision Diagram

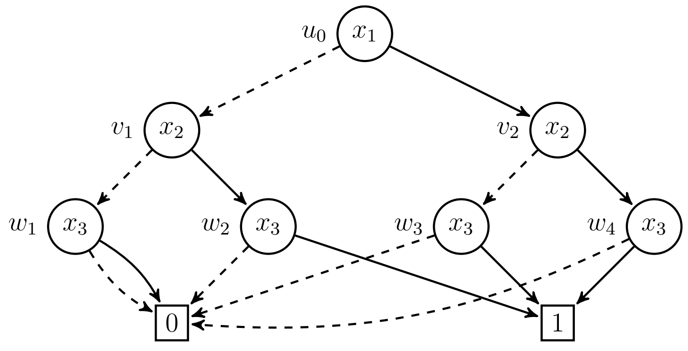
Binary decision diagrams (BDDs) are a memory-efficient data structure used to represent Boolean functions as well as to perform set-based operations. To represent the basic BDD theory employed in this work, in the following, we set $\mathbb{B} \equiv \{0, 1\}$. For any Boolean function $f \colon \mathbb{B}^n \to \mathbb{B}$ defined over n Boolean variables $X = (x_1 \ldots, x_n)$, we denote by $f|_{x_i=0}$ and $f|_{x_i=1}$ the Boolean functions that are induced from the function f by fixing the value x_i to 0, and 1, respectively. Then a BDD-based representation of f is a graph-based representation of this function that is based on Shannon expansion [109]:

$$\exists x_i \in X,\ f = (\neg x_i \wedge f|_{x_i=0}) \vee (x_i \wedge f|_{x_i=1}) \tag{4.1}$$

Definition 4.1. A binary decision diagram (BDD) is a single-rooted acyclic digraph with two types of nodes: *decision nodes* and *terminal nodes*. A terminal node can be labeled either by 0 or 1. Each decision node is labelled by a Boolean variable and it has two outgoing edges, the low edge and the high edge, with the respective edge corresponding to assigning the value of the labeling variable to 0 or to 1.

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

(a)



(b)

Figure 4.1: The truth table and BDD representations of function $f = (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$. Solid lines denote high edges and dashed lines denote low edges.

Example 4.1. As an example, Figure 4.1 illustrates a representation of the Boolean function $f = (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$ by the truth table and the BDD. The depicted BDD has two terminal nodes: terminal 0 and terminal 1. Each decision node labels a Boolean variable $x_i, i = 1, 2, 3$, and has outgoing edges directed to two children: the low edge (shown as a dashed line) corresponding to the case where the Boolean variable is assigned 0, and the high edge (shown as a solid line) corresponding to the case where the variable is assigned 1.

A path in a BDD starting from the root to the terminal 1 corresponds to a satisfying assignment of the variables for the BDD. Each variable visited by the path is assigned the value according to the outgoing edge; all other variables can take any value.

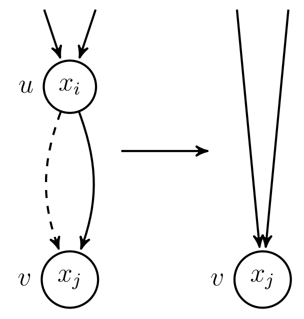
Given a BDD, it is straightforward to check whether an assignment of the variables is satisfying or not. Starting at the root, it is only necessary to follow the outgoing edge corresponding to the value of the variable in the assignment. If the terminal 1 is reached, the assignment is satisfying; if the terminal 0 is reached it is not.

Definition 4.2. An ordered binary decision diagram (OBDD) is a BDD where the variables represented by the decision nodes on any path from the root to one of the terminal nodes obey the same fixed ordering.

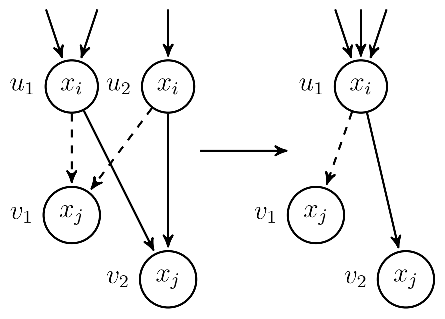
Due to the fixed variable ordering, OBDDs can be reduced by following two rules [21], which are depicted in Figure 4.2:

1. If the two successors of a node u of an OBDD when following the low and high edges reach the same node v, then u can be removed from the OBDD and all its incoming edges are redirected to node v. See Figure 4.2a.

2. If two nodes u_1 and u_2 representing the same variable reach the same two nodes when following their low and high edges, the nodes u_1 and u_2 can be merged by removing u_2 and redirecting all its incoming edges to u_1. See Figure 4.2b.



(a) Rule 1. Removal of nodes with equal successors.



(b) Rule 2. Merging of nodes with same low and high successors.

Figure 4.2: Two reduction rules for minimizing OBDDs. Solid lines denote high edges and dashed lines denote low edges.

A reduced ordered binary decision diagram (ROBDD) is an OBDD where the reduction rules are repeatedly applied until none of them can be applied any more.

An ROBDD is a canonical representation of the given Boolean function, i.e., two equivalent formulas are represented by the same ROBDD. Thanks to the reduction by which redundant internal nodes can be saved, ROBDD representations are usually memory-efficient. Throughout this work we are only concerned with ROBDDs, so whatever we write about BDDs we actually refer to ROBDDs. The size of a BDD refers to the number of decision nodes contained in this BDD.

From a computational standpoint, the power of BDDs lies in the efficiency that they provide in the execution of binary operations. Let f and f' be two Boolean functions of X. Then, it should be evident from (4.1) that a binary operator $\otimes$ between (the BDDs representing) f and f' can be recursively computed as

$$f \otimes f' = [\neg x \wedge (f|_{x=0} \otimes f'|_{x=0})] \vee [x \wedge (f|_{x=1} \otimes f'|_{x=1})] \tag{4.2}$$

where $x \in X$. The computation implied by (4.2) can have a complexity of $\mathcal{O}(|f| \cdot |f'|)$ where $|f|$ and $|f'|$ are the sizes of the BDDs representing f and f'. Note that the unary operation NOT on a BDD is simply the interchange of the 0 and 1 terminals.

A particular operator that is used extensively in the following is the *existential quantification* of a function f over its Boolean variables. For a variable $x \in X$, the existential quantification of f is defined by $\exists x.f = f|_{x=0} \vee f|_{x=1}$. Also, if $\bar{X} = (\bar{x}_1, \ldots, \bar{x}_k) \subseteq X$, then $\exists \bar{X}.f$ is a shorthand notation for $\exists \bar{x}_1.\exists \bar{x}_2. \ldots \exists \bar{x}_k.f$. In plain terms, $\exists \bar{X}.f$ denotes all those truth assignments of the variable set $X \setminus \bar{X}$ that can be extended over the set $\bar{X}$ in a way that function f is eventually satisfied.

For a more elaborate exposition of BDDs and the implementation of different fundamental operations, we refer readers to [21, 110].

4.2 Representations of DES Models

In this section, we discuss how DES models such as FA and EFA can be symbolically represented by BDDs. In particular, we will briefly introduce a number of procedures that are presented in the appended papers for symbolically representing single FA or EFA and their full synchronous compositions.

4.2.1 Characteristic Functions

Before we proceed with the discussion of these encoding procedures, we need to introduce the notion of *characteristic function*.

Definition 4.3. Let Z be a finite set so that $Z \subseteq U$, where U is the finite universal set. A characteristic function (CF) $\chi_Z : U \to \mathbb{B}$ is defined by:

$$\chi_Z(a) = \begin{cases} 1 & \text{iff } a \in Z \\ 0 & \text{otherwise.} \end{cases}$$

Since the set U is finite, in practice its elements are represented with numbers in $\mathbb{Z}_{|U|}$ or the corresponding binary x-tuples belonging to $\mathbb{B}^x$ ($x = \lceil \log_2^{|U|} \rceil$).

Set operations can be equivalently carried out on corresponding characteristic functions. For example, $Z_1 \cup Z_2$, ($Z_1, Z_2 \subseteq U$) can be mapped equivalently to $\chi_{Z_1} \vee \chi_{Z_2}$, since $Z_1 \cup Z_2 = \{\alpha \in U \mid \alpha \in Z_1 \vee \alpha \in Z_2\}$. Table 4.1 shows more operations on characteristic functions.

In the sequel, all formal discussions will be based on the corresponding characteristic functions of BDDs. Also, we will use the term BDD interchangeably with its characteristic function.

Table 4.1: Set operations as characteristic functions.

Set/Operation	Characteristic function
$\emptyset$	0
Z	1
$U - Z$	$\neg\chi_Z$
$Z_1 \cup Z_2$	$\chi_{Z_1} \vee \chi_{Z_2}$
$Z_1 \cap Z_2$	$\chi_{Z_1} \wedge \chi_{Z_2}$
$Z_1 = Z_2$	$\chi_{Z_1} \leftrightarrow \chi_{Z_2}$

4.2.2 Single Model Representation

To symbolically represent a single deterministic FA or EFA, we need two kinds of characteristic functions, which respectively characterize the state set and the transition relation. In the following, we will merely focus on the characteristic functions for a single EFA since a FA can be considered as a special case of an EFA where no variables are defined.

We start by defining different sets of Boolean variables to encode the locations, the events and the variables of an EFA E. For the encoding of the state set $Q\colon L \times D$, we employ two Boolean variable sets, denoted by X^L and $X^D = X^{D_1} \cup \ldots \cup X^{D_n}$, to respectively encode the location set L and the domain D of n variables. State $q = (\ell, v) \in Q$ is then associated with a unique satisfying assignment of the Boolean variables in $X^L \cup X^D$.

The symbolic representation of the transition relation $\rightarrow_E$ relies on the same idea but it is more complicated. Recall that a transition $\ell \overset{\sigma}{\rightarrow}_{g/a} \ell' \in\rightarrow_E$ is essentially a set of explicit transitions $(\ell, v, \sigma, \ell', v') \in \mapsto$ of EST(E), with each specifying a source-state $q = (\ell, v)$, an event σ, and a target state $q' = (\ell', v')$. Formally, we employ the variable sets X^L and X^D to encode the source state q, and a copy of X^L and X^D, denoted by $\acute{X}^L$ and $\acute{X}^D$, to encode the target state q'. In addition, we employ the Boolean variable set X^Σ to encode the alphabet of E, and we associate σ with a unique satisfying assignment of the variables in X^Σ. Then we identify the transition relation $\rightarrow_E$ of E with the characteristic function

$$\Delta(\langle q, \sigma, q' \rangle) = \begin{cases} 1 & \text{if } \ell \overset{\sigma}{\rightarrow}_{g/a} \ell' \in \rightarrow_E, v \models g, v' = a(v) \\ 0 & \text{otherwise} \end{cases}$$

That is, Δ assigns the value of 1 to $\langle q, \sigma, q' \rangle$ if there exists a transition from ℓ to ℓ' labelled by σ, the values of the variables at ℓ satisfy the guard g, i.e., $v \models g$, and the values of the variables v' at ℓ' are the result of performing action a on v.

Note that when it comes to the symbolic representation of an EFA, due to the cross-product of locations and values of variables, the resulting BDD contains unreachable states for most cases. Hence, a reachability computation might be needed to exclude all of the unreachable states in this BDD.

Example 4.2. The symbolic representation of an EFA discussed above is now exemplified on the stick-picking game introduced in Chapter 2. Figure 4.3 shows the

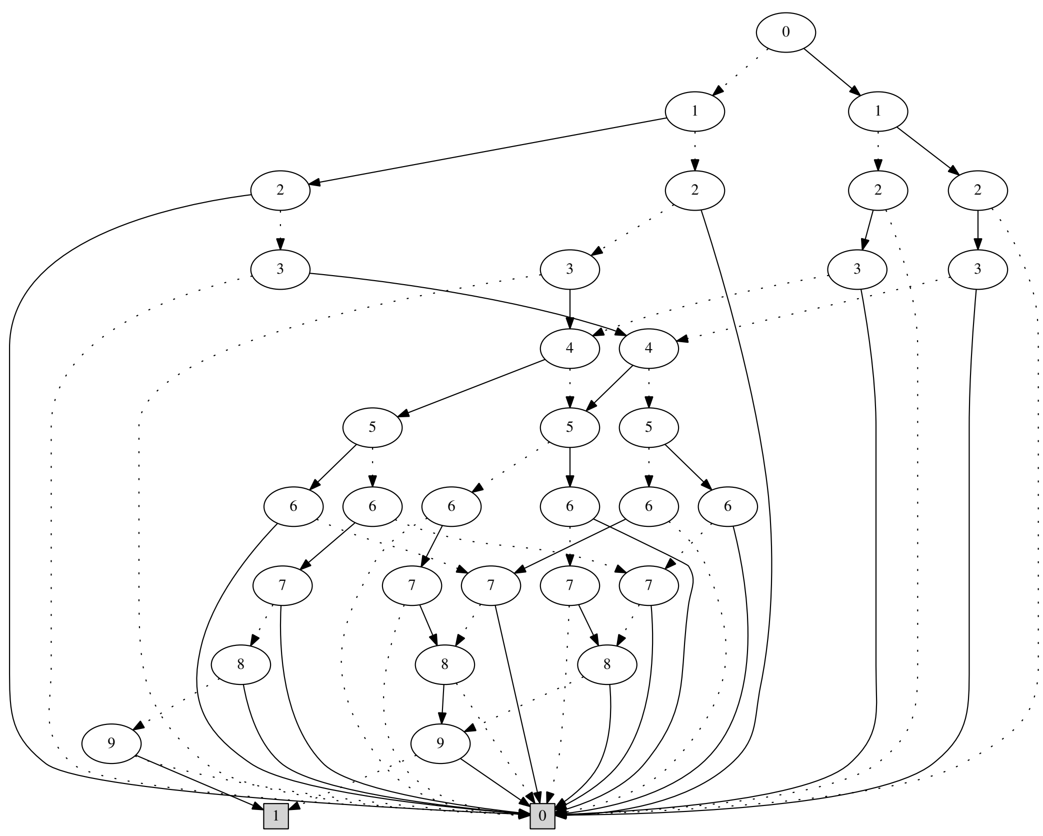


Figure 4.3: The corresponding BDD for the transition relation of the EFA in Figure 2.2.

corresponding transition relation for the EFA depicted in Figure 2.2 where each satisfying path from the root denotes an explicit state transition. Note that the BDD depicted in Figure 4.3 does not contain the cases where *sticks* < 0 and *sticks* > 5. The Boolean variables in the figure are labeled with numbers as follows.

$$\acute{X}^{sticks} = (x_9, x_8, x_7) = (9, 8, 7),\ X^{sticks} = (x_6, x_5, x_4) = (6, 5, 4),$$
$$\acute{X}^{L} = (x_3) = (3),\ X^{L} = (x_2) = (2),\ X^{\Sigma} = (x_1, x_0) = (1, 0),$$

and the encoding of locations, events, and the values of the variable *sticks* are shown in Table 4.2. As an example, we note that (011 100 1 0 00) is a true assignment. This assignment represents that the event `player1rm1`(011 100 1 0 **00**) is transited upon from the Player1 location (011 100 1 **0** 00). This transition then reaches the Player2 location (011 100 **1** 0 00), changing the *sticks* from four (011 **100** 1 0 00) to three (**011** 100 1 0 00).

As can be observed, the BDD in this example is larger than its corresponding EFA in terms of the number of nodes, however, for larger models the BDDs typically become more compact.

Table 4.2: Event, location and variable encoding for the EFA in Figure 2.2.

Event	(x_1, x_0)	Location	x_2	x_3	*sticks*	(x_6, x_5, x_4)	(x_9, x_8, x_7)
`player1rm1`	$(0, 0)$	Player1	0	1	5	$(1, 0, 1)$	$(1, 0, 1)$
`player1rm2`	$(0, 1)$	Player2	1	0	4	$(1, 0, 0)$	$(1, 0, 0)$
`player2rm1`	$(1, 0)$				3	$(0, 1, 1)$	$(0, 1, 1)$
`player2rm2`	$(1, 1)$				2	$(0, 1, 0)$	$(0, 1, 0)$
					1	$(0, 0, 1)$	$(0, 0, 1)$
					0	$(0, 0, 0)$	$(0, 0, 0)$

4.2.3 Composed Model Representation

Having a number of DES models, i.e., deterministic FA or EFA, the transition relation of the composed system can be either symbolically represented as a single BDD, or as multiple BDDs through using partitioning techniques.

Monolithic Representation

Having a set of deterministic FA, $A = \{A_1, \ldots, A_N\}$ where $N \geq 2$, the BDD representing the transition relation of $\mathbf{A} = A_1 || \ldots || A_N$ can be computed in two steps. First, we need to make all FA have the same alphabet. To this end, for each A_i where $i = 1, \ldots, N$, we attach a self-loop transition labeled with all $\sigma \in \Sigma_{\mathbf{A}} \backslash \Sigma_{A_i}$ to all states of A_i. We then compute the characteristic function of $\mathbf{A}$ by performing a logical AND on all characteristic functions that represent the transition relations of $A_1, \ldots, A_N$, i.e., $\Delta_{\mathbf{A}} = \bigwedge_{i=1}^{N} \Delta_{A_i}$.

Given a model containing a set of EFA, $E = \{E_1, \ldots, E_N\}$, the BDD representing the transition relation of $\mathbf{E} = E_1 || \ldots || E_N$, however, cannot be obtained by simply conjuncting the BDDs representing the transition relations of all the included EFA, due to those non-updated variables. If there exists a variable that is not updated for a transition in the composed model $\mathbf{E}$, the nodes corresponding to the Boolean variables representing the updated values of this variable will not be present in the BDD. Logically, this means that after the transition is taken, the value of the variable can be any value in its domain. However, by Definition 2.4, it is the current value of the variable that should remain after the transition is taken. On the other hand, one might wonder why not consider encoding this “current” value when constructing the characteristic functions for each individual EFA. However, the problem is that whether a variable is updated for a transition labeled with an event cannot be determined until all the transitions in $E_1, \ldots, E_N$ labeled by the same event are considered.

In [38], the authors proposed an approach for constructing the monolithic BDD representing the transition relation of $\mathbf{E} = E_1 || \ldots || E_N$ correctly, by taking into account the updates of variables in all the components simultaneously. Briefly speaking, the approach computes the characteristic function (CF) of $\mathbf{E}$ in three steps:

1. Compute a CF, representing the transition relation of $\mathbf{E}$ but without including the Boolean variables for the updated values of variables.

2. Compute a CF, representing the set of transitions in $\mathbf{E}$ where the variables are updated. In addition, compute another CF to represent all the transitions where variables are not updated, and then perform a number of symbolic computations on this CF to make those non-updated variables update to the previous values (i.e., the source values). The resulting CF is obtained by performing the OR operation on these two CFs.

3. Compute the CF representing the transition relation of $\mathbf{E}$ by conjuncting the CFs computed from Step (1) and (2).

We refer readers to [38] for details of this encoding scheme.

Disjunctive Representation

Alternatively, the transition relation of the composed system can be symbolically represented as a set of BDDs through the adoption of either the *disjunctive* partitioning technique or the *conjunctive* partitioning technique introduced in [67, 68]. In [Paper 1], these two techniques are evaluated on a set of benchmark examples. In this work, we merely focus on two variants of the disjunctive partitioning technique.

For deterministic FA models, the *automaton-based partitioning* approach that is introduced and formulated respectively in [111] and [Paper 1] can be used to represent the transition relation of the composed automaton. More specifically, let $A = \{A_1, \ldots, A_N\}$ where $N \geq 2$ and $\Delta_{A_1}, \ldots, \Delta_{A_N}$ be the corresponding BDDs representing the transition relations of $A_1, \ldots, A_N$. For each A_i, the automaton-based partitioning approach constructs a BDD for A_i, representing the set of composed transitions labeled with all $\sigma \in \Sigma_{A_i}$ in $\mathbf{A} = A_1 || \ldots || A_N$ according to the following steps:

1. Identify all the automata $A_j \in A$ that interact with $A_i \in A$ through any common event. We shall denote by $\mathbf{D}(A_i)$ the set of the automata that interact with A_i. Hence,
$$\mathbf{D}(A_i) = \{A_j \in A \mid i \neq j \text{ and } \Sigma_{A_i} \cap \Sigma_{A_j} \neq \emptyset\}.$$
Then we construct a CF by performing the AND operation on all Δ_{A_j}.

2. For all $A_j \in \mathbf{D}(A_i)$, construct the CF representing the transition set $\{(q, \sigma, q') \mid q, q' \in Q_{A_j} \text{ and } \sigma \in (\Sigma_{A_i} \backslash \Sigma_{A_j})\}$.

3. For all $A_k \in A \backslash \mathbf{D}(A_j)$ where $A_k \neq A_i$, construct the CF representing the transition set $\{(q, \sigma, q') \mid q, q' \in Q_{A_k} \text{ and } \sigma \in \Sigma_{A_i}\}$.

4. Compute the target CF for A_i by utilizing the CFs computed from Step (1–3).

The computations obtaining the aforementioned CFs are detailed in [Paper 1].

The partitioning representation of the transition relation of a composed EFA can be symbolically computed by applying the *event-based partitioning* approach introduced in [Paper 2]. Let $E = \{E_1, \ldots, E_N\}$ be a set of $N \geq 2$ EFA defined over the variable set $v = (v_1, \ldots, v_n)$. For each event $\sigma \in \Sigma_{\mathbf{E}}$, the event-based partitioning approach constructs a BDD for σ, representing all the transitions in $\mathbf{E} = E_1 || \ldots || E_N$ that are all labeled by event σ with the following steps:

1. Identify all the EFA, denoted by the set E^σ, that contain the event σ in the alphabets. That is, $E^\sigma = \{E_i \in E \mid \sigma \in \Sigma_i\}$.
2. Compute a CF, representing the set of transitions in $\mathbf{E}^\sigma = ||_{E \in E^\sigma} E$ that are all labeled with the event σ. The updating and non-updating of variables for different composed transitions in $\mathbf{E}^\sigma$ are symbolically handled in this step. We refer the reader to [Paper 2] for the detailed computations.
3. Compute a set of CFs, with each one representing the transition set $\{(q, \sigma, q') \mid q, q' \in E_i \text{ where } E_i \in E \backslash E^\sigma \text{ and } q = q'\}$.
4. Compute the target CF for all events $\sigma \in (\Sigma_1 \cup \ldots \cup \Sigma_N)$ by conjuncting the CF computed in Step (2) and the set of CFs computed in Step 3.

4.3 Symbolic State-Space Exploration

Exploring state-spaces is the most fundamental and challenging task in the supervisor synthesis. After representing DES models as BDDs, as shown in Section 4.3.1, the state-space explorations can be carried out on BDDs with the provided set-based symbolic operations.

However, a straightforward transformation from the explicit state-space enumeration methods into the BDD-based computation scheme does not guarantee that synthesis will become remarkably efficient. This is because for the BDD-based algorithms, the computational complexity is no longer dependent on the number of states, but on the sizes of the intermediate BDDs that are constructed for storing the explored states and transitions. As the intermediate BDDs are constructed during the state-space exploration, a dramatic increase in the number of BDD nodes to represent particular state or transition sets might occur.

In Section 4.3.2 and 4.3.3, we briefly report two symbolic techniques for the efficient state-space exploration with BDDs. In particular, for general DES, the first technique enforces a structure for exploring the sought states on the underlying DES state-space that is represented as a set of partitioned BDDs, in order to keep the intermediate BDDs as small as possible. The formal formulations of this technique are respectively presented in [Paper 1] and [Paper 2] for deterministic FA and EFA models. By taking the advantage of some structural characterizations regarding the RAS safety, the second technique focuses on the computation of boundary unsafe states of the RAS-modeling EFA while avoiding the compete exploration of state-spaces.

The detailed exposition of this technique can be found in [Paper 3]. In [Paper 4], we combine these symbolic techniques for a more efficient computation of boundary unsafe states in complex RAS.

4.3.1 Symbolic Safe State Synthesis

The safe-state-synthesis algorithm introduced in Chapter 2 can be transformed into the corresponding symbolic version in a straightforward way. Having the BDD representing the transition relation of $S_0 = G||K$, denoted by Δ_{S_0}, and the BDDs χ_{Q^m} and χ_{Q^x} that represent the set of marked states Q^m and the set of the forbidden states Q^x, Algorithm 5 depicts the symbolic operations for computing all the states that are coreachable to states in Q^m and also do not belong to the set Q^x.

```
Algorithm 5: Symbolic-Restricted-Backward
  Input: $\chi_{Q^m}$, $\chi_{Q^x}$ and $\Delta_{S_0}$
  Output: The BDD representing all the states in $S_0$ that are coreachable to
          $Q^m$ and not in $Q^x$.
1 $i := 0, \chi_{Q_0} := \chi_{Q^m} \wedge \neg\chi_{Q^x}$;
2 repeat
3     $i := i + 1$;
4     $\chi_{Q_i} := \chi_{Q_{i-1}} \vee preImage(\Delta_{S_0}, \chi_{Q_{i-1}}[X^Q \to \acute{X}^Q]) \wedge \neg\chi_{Q^x}$;
5 until $\chi_{Q_i} \leftrightarrow \chi_{Q_{i-1}}$;
6 return $\chi_{Q_i}$;
```

In Line 4 of Algorithm 5, the *preImage* operator is defined as follows,

$$preImage(\Delta_{S_0}, \chi_Q) = \exists \acute{X}^Q \cup X^{\Sigma_{S_0}}.\ (\Delta_{S_0} \wedge \chi_Q).$$

The *preImage* operator takes as input the BDDs representing a transition relation Δ_{S_0} and a set of states χ_Q, and outputs a BDD representing all the states that are one-step coreachable to χ_Q. For FA models, $\acute{X}^Q$ denotes the set of Boolean variables for encoding target states; for EFA models, $\acute{X}^Q$ denotes the union of Boolean variables for encoding locations and domains of variables, i.e., $\acute{X}^Q = \acute{X}^L \cup \acute{X}^D$. Note that when applying the *preImage* operation for computing more coreachable states, the state set χ_{Q_i} needs to be represented by the target Boolean variables of $\acute{X}^Q$. To this end, the operation $[X^Q \to \acute{X}^Q]$ in Line 4 is used to denote the replacement of all Boolean variables of X^Q by those of $\acute{X}^Q$, so that the coreachable states identified at each iteration are eventually represented by X^Q and the backward search can continue.

Example 4.3. To give a flavor of how Algorithm 5 works, we apply it to the BDD-based representation of the stick-picking game introduced in Chapter 2 and describe the first iteration of computations for expanding the coreachable state set. Please refer to Figure 4.4 throughout the following demonstration.

To symbolically compute the coreachable states, Algorithm 5 starts with the marked state (Player2, Stick0) in Figure 4.4. This state is represented as $(\cdots 000 \cdot 1 \cdot\cdot)$

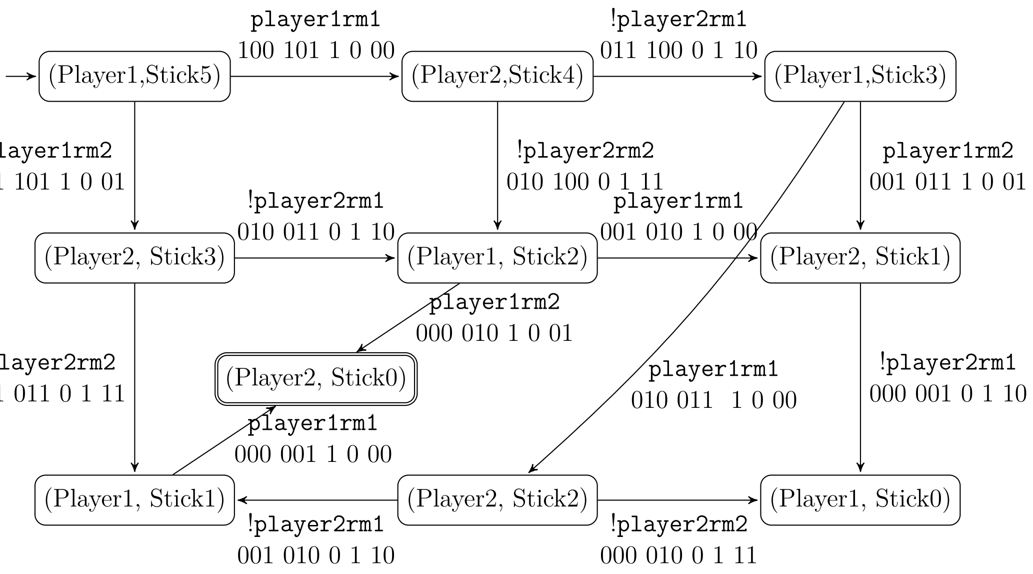


Figure 4.4: The supervisor candidate S_0 = Player||Stick where all the unreachable states are omitted for simplicity. Events prefixed with the exclamation mark (!) are uncontrollable events. The binary numbers are the respective transition encodings that correspond to the true assignments of the BDD in Figure 4.3.

that indicates the vector of values assigned to the Boolean variables $x_9, \ldots, x_0$, where the $\cdot$ sign denotes the "don't care". As depicted in Line 4 of Algorithm 5, the symbolic operation first assigns the values 0001 to the Boolean variables x_9, x_8, x_7, x_3 that are used for encoding the target states while the values of x_6, x_5, x_4, x_2 become the don't-care. That is, the encoding of the marked state becomes $(000 \;\cdots\; 1 \;\cdot\; \cdot\cdot)$ presenting that the target number of *sticks* is 0, and the target location is Player2. This expression, represented as a BDD, is conjuncted with the BDD for the transition relation depicted in Figure 4.3, and then satisfying assignments to the variables corresponding to the don't-care are found. The operation results in two transitions, (000 001 1 0 00) and (000 010 1 0 01) that are shown in Figure 4.4. These transitions represent that the number of sticks goes from 1 to 0, and 2 to 0, respectively, as Player1 takes 1 and 2 sticks, moving from the location Player1 to location Player2, on the respective events `player1rm1` and `player1rm2`.

Next, Algorithm 5 needs to identify the source states of these two transitions. So the target state and event bits are marked as don't-care, giving $(\cdots\; 001 \;\cdot\; 0 \;\cdot\cdot)$ and $(\cdots\; 010 \;\cdot\; 0 \;\cdot\cdot)$. These bit-vectors represent the two states (Player1, Stick1) and (Player1, Stick2), that later are collected into the χ_{Q_i}. The algorithm continues to iterate over the coreachable states, until no more coreachable states are found.

The symbolic versions of the restricted forward search (Algorithm 4) and the uncontrollable backward search (Algorithm 3) of the safe-state-synthesis algorithm can be similarly transformed, and thus, they are omitted.

4.3.2 State-Space Exploration on Partitioned BDDs

It can be observed from Algorithm 5 that the basic symbolic algorithm for computing reachable or coreachable states requires a single BDD representing the monolithic transition relation of S_0. However, for many practical applications, the size of this BDD might be too large to be constructed. Moreover, since the computational complexity for BDD-based symbolic computation is no longer dependent on the number of states but on the number of nodes in the BDDs, performing the breadth-first search on this entire BDD might lead to a huge number of nodes in the intermediate BDDs during the state-space exploration.

Through employing the disjunctive partitioning techniques, as explained in Section 4.2.3, the transition relation of S_0 can be symbolically represented as a set of smaller BDDs that represent a set of partial transition relations, with each one corresponding to a particular finite automaton or event. Having a disjunctive representation of S_0, collectively denoted by $\Delta_1, \ldots, \Delta_n, n \geq 2$, the *preImage* operation depicted in Algorithm 5 can be changed accordingly as follows.

$$\begin{aligned}
preImage(\Delta_{S_0}, \chi_Q) &= \exists \acute{X}^Q \cup X^{\Sigma_{S_0}}.\ (\Delta_{S_0} \wedge \chi_Q) \\
&= \exists \acute{X}^Q \cup X^{\Sigma_{S_0}}.\ \big((\Delta_1 \vee \ldots \vee \Delta_n) \wedge \chi_Q\big) \\
&= \exists \acute{X}^Q \cup X^{\Sigma_{S_0}}.\ \big((\Delta_1 \wedge \chi_Q) \vee \ldots \vee (\Delta_n \wedge \chi_Q)\big) \\
&= \big(\exists \acute{X}^Q \cup X^{\Sigma_{S_0}}.\ (\Delta_1 \wedge \chi_Q)\big) \vee \ldots \vee \big(\exists \acute{X}^Q \cup X^{\Sigma_{S_0}}.\ (\Delta_n \wedge \chi_Q)\big)
\end{aligned} \tag{4.3}$$

That is, instead of manipulating one monolithic BDD, we can identify the set of coreachable states from the state-space by iterating each partial transition relation, that, in most cases, is substantially smaller than the monolithic BDD Δ_{S_0}.

The adoption of the disjunctive partitioning technique and the updated *preImage* operation described in (4.3), however, does not address the problem of the large sizes of intermediate BDDs during the state-space exploration. The updated *preImage* essentially performs a breadth-first search on the set of partial transition relations. This symbolic computation scheme identifies the coreachable states in an almost random manner, which prevents the BDD reductions being applied in the intermediate BDDs to the maximal extent. Therefore, sizes of these relevant BDDs might become extremely large, causing the state-space explosion.

To reach significant BDD reduction it is crucial to explore the search space in a structured way. In [Paper 1] and [Paper 2], we proposed two symbolic algorithms for the efficient state-space exploration. The *workset* algorithm of [Paper 1] is a variant and an extension of the algorithm of [34] and [66] that is based on the automaton-based representation of the state-space of DES modeled by FA. The *extended workset* algorithm proposed in [Paper 2], on the other hand, is an event-based partitioning algorithm that is applicable to both FA and EFA models. More specifically, taking as input the BDD representing the initial state of S_0 and the set of BDDs representing the set of partial transition relations, the extended workset algorithm maintains a set of active partial transition relations during the execution. For each iteration, the partial transition relation where new states are most likely to be identified by the existing states, is selected for the exploration. If more states are found, some transition relations that are relevant to the chosen one are appended in the workset for further exploration. The algorithm terminates when there is no partial transition relation left in the workset.

For a more detailed exposition of these two symbolic algorithms and the proof of correctness, refer to [Paper 1] and [Paper 2].

4.3.3 Partial State-Space Exploration for RAS

In comparison with general DES, RAS possess some special structural characteristics that can be utilized to design the maximally permissive DAP more efficiently. In [Paper 4] and [Paper 5], we have a symbolic framework for this purpose. More specifically, by modeling any given RAS instance as EFA, the proposed framework employs several BDD-based algorithms for symbolically computing the target DAP. Besides the employment of symbolic computation, additional efficiencies for the algorithms are obtained from the fact that they avoid the complete exploration of the underlying RAS state-space. This capability is established upon the crucial fact that, in the considered RAS dynamics, unsafety is defined by inevitable absorption into the system deadlocks. Therefore, the target unsafe states can be retrieved by a computation that starts from the RAS deadlocks and "backtraces" the RAS state-space until it hits the boundary between the safe and unsafe subspaces. Furthermore, the entire set of boundary unsafe states can be effectively represented by its minimal elements since the notion of unsafety presents a monotonicity property that endows this set with properties similar to those of a right-closed set [80]. In this section, we illustrate the two symbolic algorithms of [Paper 3], by computing the DAP for the RAS instance introduced in Example 3.1 of Chapter 3.

In the sequel, these computations are illustrated on the EST model in Figure 4.5. The depicted EST model includes only the RAS feasible states that are reachable from the initial and target state $\mathbf{s}_0$. In the approaches presented in [Paper 3] and [Paper 4], this depicted EST, together with other unreachable states and infeasible states, is symbolically represented by BDDs. For reasons that will become clear in the following, it is pertinent to assume that the BDD that represents the transition relation of EST($\mathbf{E}$), denoted by $\Delta_{\mathbf{E}}$, is partitioned into two BDDs Δ_L and Δ_A that collect respectively the transitions in $\Delta_{\mathbf{E}}$ corresponding to the transitions labeled by the loading events, and the transitions labeled by the process-advancing events.

Identification of the feasible deadlock states The deadlock states pursued in this work are those states from where no process-advancing events can occur. With $\Delta_{\mathbf{E}}$ available, this set of states can be symbolically computed as follows. First, we collect the set of all the target states of the transitions in $\Delta_{\mathbf{E}}$ and denote it as the set T; obviously, T contains all the states $s_i, i = 1, ..., 11$, but it also contains other infeasible states. Then, we collect the set of all the source states of the transitions in Δ_A and denote it as the set E, i.e., $E = \{\mathbf{s_1}, \mathbf{s_2}, \mathbf{s_3}, \mathbf{s_4}, \mathbf{s_5}, \mathbf{s_6}, \mathbf{s_9}\}$. The reader should note that states $\mathbf{s_0}, \mathbf{s_7}, \mathbf{s_8} \notin E$, since none of the relevant transitions

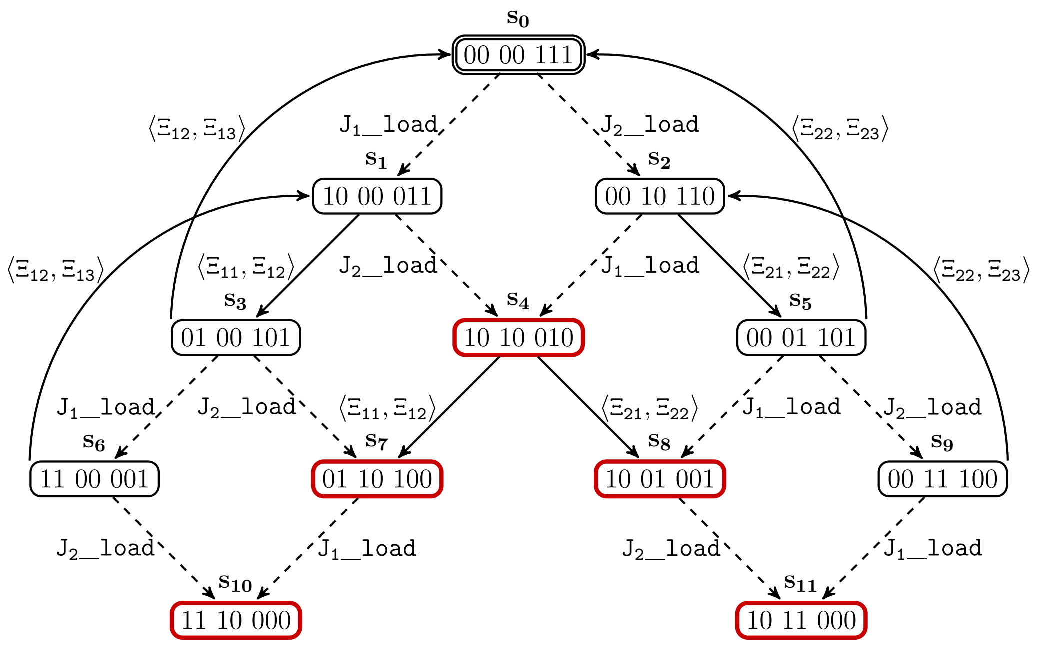


Figure 4.5: The EST model of the EFA $\mathbf{E}$ that is composed of the two EFA $E(J_1)$ and $E(J_2)$ depicted in Figure 3.3. In the depicted EST, solid lines denote the loading transitions while dashed lines denote the process-advancing transitions. States depicted in red are unsafe.

$\langle \mathbf{s_0}, \mathtt{J_1\_load}, \mathbf{s_1}\rangle, \langle \mathbf{s_0}, \mathtt{J_2\_load}, \mathbf{s_2}\rangle, \langle \mathbf{s_7}, \mathtt{J_1\_load}, \mathbf{s_{10}}\rangle$ and $\langle \mathbf{s_8}, \mathtt{J_2\_load}, \mathbf{s_{11}}\rangle$ belongs to Δ_A. The set of deadlock states, D, is obtained by removing from set T the initial state $\mathbf{s_0}$, and the states belonging to the set E.

Since the set D is computed from the entire set of transitions that is contained in $\Delta_{\mathbf{E}}$, it might contain deadlock states that are infeasible (i.e., they violate the constraints of (3.2)). The presence of these infeasible states in D would increase unnecessarily the computational cost of the second stage. Hence, as the last step, the symbolic representation of D, χ_D, is filtered through its conjunction with the BDD χ_F that encodes the constraints (3.2), in order to obtain the set of feasible deadlock states, FD. For the EST shown in Figure 4.5, $FD = \{\mathbf{s_7}, \mathbf{s_8}, \mathbf{s_{10}}, \mathbf{s_{11}}\}$.

Identification of the feasible boundary unsafe states Having obtained the set FD of the feasible deadlock states, the algorithm proceeds with the symbolic computation of the feasible boundary unsafe state set, denoted by FB. We employ the set U to collect all the identified unsafe states. At each iteration, the set U_{new} defines the set of the unsafe states that are to be processed at that iteration, through one-step-backtracking in $\Delta_{\mathbf{E}}$ in an effort to reach and explore new states. Both U and U_{new} are initialized to FD.

We start with the extraction of all states that can be reached from U_{new} by backtracing some transitions in Δ_A. We shall denote the set of extracted states and transitions respectively by the sets $S\hat{U}$ and $\Delta_{\hat{U}}$. With respect to the EST depicted in in Figure 4.5, the state $\mathbf{s_4} \in S\hat{U}$ and $\Delta_{\hat{U}} = \{(\mathbf{s_4}, \langle \Xi_{11}, \Xi_{12}\rangle, \mathbf{s_7}), (\mathbf{s_4}, \langle \Xi_{21}, \Xi_{22}\rangle, \mathbf{s_8})\}$.

We then perform a one-step forward search over Δ_A with the states in $S\hat{U}$ and collect all the transitions of Δ_A with the source states belonging to $S\hat{U}$. The set of these identified and collected transitions is denoted by Δ_{SA}; clearly, $\Delta_{SA} = \{(\mathbf{s_4}, \langle \Xi_{11}, \Xi_{12}\rangle, \mathbf{s_7}), (\mathbf{s_4}, \langle \Xi_{21}, \Xi_{22}\rangle, \mathbf{s_8})\}$ in the considered EST. By removing from Δ_{SA} all the transitions belonging to $\Delta_{\hat{U}}$[1] and extracting the corresponding source states, we can identify all the states that are not unsafe states at the current iteration. Moreover, if we remove these states from $S\hat{U}$, new unsafe states at the current iteration can be obtained. In the considered EST, since $\Delta_{SA} = \Delta_{\hat{U}}$, the state $\mathbf{s_4}$ is a newly identified unsafe state. Hence, at the end of the current iteration, $U = \{\mathbf{s_4}, \mathbf{s_7}, \mathbf{s_8}, \mathbf{s_{10}}, \mathbf{s_{11}}\}$, and for the next iteration of the search process, $U_{new} = \{\mathbf{s_4}\}$. The backward search process terminates after the second iteration, since no unsafe state can be identified from the state $\mathbf{s_4}$. At this point, the symbolic approach proceeds to extract the boundary states from the state set U. For that, the symbolic computations extract from $\Delta_{\mathbf{E}}$ all the transitions with the target states belonging to the states in U; the relevant transition set is denoted by $\Delta_{\mathcal{B}}$. Next, the algorithm retrieves from $\Delta_{\mathcal{B}}$ the transition set $\Delta_{S\mathcal{B}}$, where the source states of the included transitions are safe states. Finally, the boundary unsafe state set FB is obtained by extracting the target states from $\Delta_{S\mathcal{B}}$. For the depicted EST, after performing the aforementioned operations, we have $FB = \{\mathbf{s_4}, \mathbf{s_7}, \mathbf{s_8}, \mathbf{s_{10}}, \mathbf{s_{11}}\}$.

Identification of the minimal boundary unsafe states. An important implication of the invariants of (3.2) is that, at any feasible state of the underlying EFA state-space, the values of the resource variables can be induced from the values of the instance variables. In other words, it is sufficient to have the specification of its process variables to uniquely determine the feasible state s of the considered EST. Hence, one can obtain a more compact symbolic representation of the set of feasible boundary unsafe states, χ_{FB}, by eliminating from the elements of χ_{FB} the values that correspond to the resource variables. Letting X^R and X^I respectively denote the Boolean variables representing the values of the resource variables $vr_i,\ i = 1, 2, 3$, this elimination can be performed through the following existential quantification:

$$\chi_{FB} := \exists(X^R \cup X^I).\ \chi_{FB}. \tag{4.4}$$

For the considered example, the state set FB that is returned by the operation of Eq. (4.4) can be represented as follows:

$$FB = \{1010(\mathbf{s_4}), 0110(\mathbf{s_7}), 1001(\mathbf{s_8}), 1110(\mathbf{s_{10}}), 1011(\mathbf{s_{11}})\}.$$

Given any two feasible boundary unsafe states $\mathbf{s}$, $\mathbf{s}'$ represented according to the logic of (4.4), we consider the ordering relation "$\leq$" on them that is defined by the application of this relation componentwise; i.e.,

$$\mathbf{s} \leq \mathbf{s}' \iff (\forall k = 1, \ldots, K, \mathbf{s}[k] \leq \mathbf{s}'[k]), \tag{4.5}$$

[1]In the general case, we also need to remove from Δ_{SA} the transitions that were identified and collected into $\Delta_{\hat{U}}$ from the earlier iterations.

where $\mathbf{s}[k]$ and $\mathbf{s}'[k]$ are the values of the k-th instance variable for $\mathbf{s}$ and $\mathbf{s}'$. Furthermore, we use the notation '$<$' to denote that the condition (4.5) holds as strict inequality for at least one component $v_k \in \{v_1, \ldots, v_K\}$. It is shown in [80] that if state $\mathbf{s}$ is unsafe and state $\mathbf{s}'$ satisfies $\mathbf{s} \leq \mathbf{s}'$, then the state $\mathbf{s}'$ is also unsafe. Hence, under the state representation of (4.4), the set FB can be effectively defined by the subset of its minimal elements. We shall denote this subset by $\overline{FB}$, i.e., $\overline{FB} \equiv \{\mathbf{s} \in FB \mid \nexists \mathbf{s}' \in FB \text{ s.t. } \mathbf{s}' < \mathbf{s}\}$. A symbolic algorithm for the computation of $\overline{FB}$ from FB is provided in [Paper 3]. We also note, for completeness, that in the considered example, $\overline{FB} = \{\mathbf{s_4}, \mathbf{s_7}, \mathbf{s_8}\}$.

4.4 Experimental Results

The BDD-based symbolic algorithms for representing and exploring state-spaces of DES models that are briefly discussed in Section 4.2 and Section 4.3, respectively, have been implemented and integrated into the software tool Supremica [23, 24, 25]. In this section, we discuss the experimental results obtained by applying these implemented algorithms on a set of academic and industrial examples. For the detailed discussions, we refer readers to the appended papers.

Table 1 of [Paper 1] shows the comparison between the presented workset algorithm and one existing symbolic algorithm that uses the conjunctive partitioning technique to represent and explore state-spaces. The benchmark examples used in the experiments are all modeled as deterministic FA. As can be seen from the results reported in the table, the workset algorithm clearly demonstrates better computational efficiency in terms of both time and memory usage, compared to the other symbolic algorithm. Moreover, Table 2 of [Paper 1] compares the computation time of the workset algorithm using different heuristics for selecting partial transition relations during the execution of the algorithm.

In [Paper 2], the comparison is made between the symbolic algorithm presented in [38] and the extended workset algorithm by applying them on the set of DES examples that are modeled as EFA. It can be observed from Table 3 of [Paper 2] that both the algorithm of [38] and the extended workset algorithm can handle a number of relatively large examples and synthesize the supervisors in a short time. However, the extended workset algorithm exhibits better scalability comparing to the algorithm of [38], as indicated by the maximal size of the intermediate BDDs constructed in the state-space exploration. The data obtained in this comparison also confirms the statement that the complexity of symbolic computation using BDDs is not dependent on the number of states, but on the sizes of intermediate BDDs.

In [Paper 3], we report the results from a series of computational experiments in which the proposed algorithm is applied on a number of randomly generated instantiations of different RAS classes. The comparison of the computation time and the maximal memory usage is made between the proposed algorithm and a variant of the symbolic algorithm presented in [38]. As shown in Table 1 of [Paper 3], by taking advantage of the particular structure properties, the proposed algorithm avoids the full exploration of the state-spaces of the RAS instances. Hence, the proposed algorithm requires fewer iterations to compute the target unsafe states than the existing approach, and it tends to have better computation time. Furthermore, the avoidance of the exploration of the whole RAS state-space enables less memory during its execution, especially for RAS instances with small unsafe state regions. Also, [Paper 3] compares the computation time of the proposed algorithm to the computation time of the algorithm presented in [105]. The obtained results reveal that for the RAS instances with simple linear process flows and simple or conjunctive resource allocation, the proposed algorithm outperforms the algorithm of [105]. Some of the largest cases suggest that the gains attained by the symbolic algorithm can be up to two orders of magnitude faster. On the other hand, for RAS instances possessing routing flexibility, the algorithm of [105] is competitive to the symbolic algorithm.

The symbolic algorithm presented in [Paper 3] for computing the boundary unsafe states is carried out on the monolithic symbolic representation of the underlying RAS state-space. In [Paper 4], an attractive symbolic algorithm is presented for computing the target critical states from multiple simpler BDDs rather than a single monolithic one. Table 1 of [Paper 4] reports the experimental results of applying this symbolic algorithm to the same set of RAS instances used in [Paper 3]. The perusal of the data shown in the table reveals that the proposed algorithm is more efficient in terms of its memory requirements compared to the symbolic algorithm of [Paper 3], for all the tested RAS instances. The computation time is also improved for most of the RAS instances with simple linear process flows and simple or conjunctive resource allocation.

Chapter 5

Case Study

In this chapter, an industrial example is presented to illustrate the EFA modeling framework and the symbolic approaches that are introduced in [Paper 1] and [Paper 2]. The case study is based on an example introduced in [112] where it was used to compare different formal approaches to develop control logic for coordinating the manufacturing systems. This chapter is based on the book chapter [113] where the author of this thesis was responsible for developing the EFA model and applying the proposed symbolic approaches to compute and represent the supervisor. In [Paper 5], another case study is represented to demonstrate the applicability of the proposed approaches of [Paper 3] and [Paper 4] to the elimination of deadlocks in multithreaded software.

5.1 The Manufacturing System

The book [112] systematically presents and compares 18 different approaches to the control of a real-world production cell. The example has also been treated by other researches, among them [114, 115]. The system models part of an actual metal-processing plant, where metal blanks are fed into the system on a feed belt, are picked up by a robot that places the blanks in a press, and when the metal blank has been pressed, the same robot picks it out of the press and puts it on a deposit belt.

The description of the process largely follows [112] and [114]. However, our example will include the test unit that is described in [115], and assume that there are at most two blanks on the belts.

5.1.1 The Plant

The system involves seven manufacturing devices set up in a specific configuration, see Figure 5.1. The devices are a feed belt, a rotating elevating table, a two-armed robot, a press, a deposit belt, a traveling crane, and a test unit.

The feed belt is a conveyor belt that can transport metal blanks from west to east. When the motor is turned on, the belt moves eastward transporting metal blanks placed on it. A binary sensor at the eastern side outputs a logical 1 when a

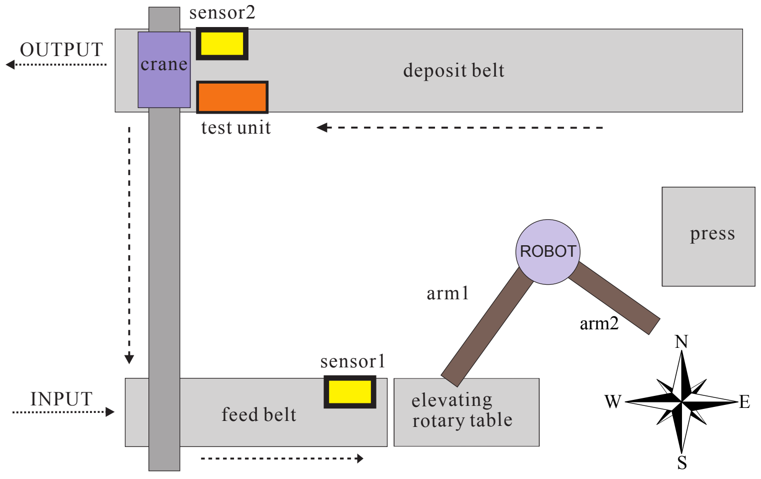


Figure 5.1: Layout of the example cell. Note that the initial system state is depicted. The arrows depict the flow of metal blanks.

metal blank is detected. The feed belt can be loaded with metal blanks from either the traveling crane, or from the input.

The rotary elevating table is basically a buffer of size one. It can rotate to be in line with the feed belt, or to be in a position for a metal blank to be picked by the robot. The table can also move up and down between two positions. The lower position is aligned with the feed belt, and the upper position is aligned with the robot.

The robot is fitted with two arms, arm_1 and arm_2, see Figure 5.1. These arms are placed on different heights and are not vertically movable. At the end of each arm there is an electromagnet. When activated, the magnet picks up any metal blank in the close vicinity, and when deactivated any metal blank held will be dropped. The robot can rotate arbitrarily, both clockwise and counter-clockwise. Extraction and detraction of arm_1 and proper rotation allow the robot to pick metal blanks from the rotating elevating table, and to load the press. Similarly, arm_2 allows the robot to unload the press, and also to drop blanks at the deposit belt.

The press consists of two horizontal plates, between which it forges metal blanks by pressing the lower plate up against the upper plate. To be loaded with a blank, the press has to lower its plate to a middle position, and to be unloaded, the press has to lower its plate to the bottom position.

The deposit belt transports metal blanks from east to west, see Figure 5.1. Similarly to the feed belt, there is a sensor at the far west end that detects metal blanks. And similarly to the feed belt, and as assumed by [114], the deposit belt has room for only two metal blanks.

At the end of the deposit belt, there is a test unit which, when the deposit belt sensor is activated, determines whether the metal blank was forged correctly. The test unit emits a logical 1 if the metal blank has been correctly forged, and a logical 0 if not. The test unit requires the deposit belt to be stopped when a metal blank is detected at the sensor, otherwise it does not have ample time for its assessment.

Furthermore, the deposit belt is unloaded either to the external storage or by the traveling crane. A metal blank sensed at the sensor is moved to the external storage by simply running the belt until the blank is no longer detected by the sensor.

The traveling crane can pick up metal blanks from the deposit belt and transport them to the feed belt. To pick up blanks, the deposit belt has to be stopped, but the traveling crane can drop blanks on the feed belt even if it moves.

5.1.2 The Specification

For the system to function properly we need to impose a number of requirements. These are both safety requirements and liveness requirements. These requirements will correspond to the controllability and non-blocking properties of the supervisor, respectively.

As described by [112], the safety requirements concern four basic principles:

- Limitations on machine mobility; the robot, rotating table, press, and traveling crane must not move too far, otherwise they may be damaged.

- Collisions avoidance; when extracting its arms, the robot must not collide with the press.

- Not drop metal blanks on the shop floor; blanks must be handled so that they are always held by some device.

- Sufficient separation of metal blanks on the belts; for the sensors to be able to distinguish between the metal blanks, and to be able to stop before a metal blank falls off at the end of a belt, metal blanks have to be separated by some minimal distance.

Some of these requirements will be taken care of by the assumed inherent behavior of the plant, while others need to formulated as specifications so that the synthesized supervisor will guarantee that they are upheld.

5.2 The Model

A typical manufacturing system comprises a set of interacting manufacturing *devices* arranged in some specific *configuration*. In the example cell, the devices are the robot, the belts, press etc, and the configuration relates to the particular layout. Inherent device functionality arises due to the specific level of “intelligence” of a device, meaning that the actions that it can perform are more or less low-level. Also, inherent system functionality arises due to the configuration; two devices may interact only if their physical boundaries are aligned. In the example cell, the feed belt can interact with the crane and the rotating table, but not directly with any other device, simply due to the layout of the cell.

The level of “intelligence” of the devices naturally affects what can be considered as plant and what can be considered as specification. Typical for real-life industrial systems is that there are low-level safety functionality implemented in code that guarantees that the devices do not break themselves. Industrial devices are simply too costly not to have this low-level safety functionality implemented. For instance, movable devices are typically equipped with safety mechanisms that prevent their motors from continuing to run, and hence risk burning or otherwise breaking, at an end-point. Thus, this safety requirement is naturally integrated into the plant model.

On the other hand, safety requirements that concern the interaction between two devices are typically not implemented as part of the physical plant. Therefore, it is not reasonable to model such requirements as part of the plant, but they must be regarded as specification. If the plant models a system where no devices can collide, for instance, then the specification cannot require avoidance of any states where the devices do collide.

However, there exists behavior that can be either modeled as plant, or specification. For instance, in the example cell there is a requirement that blanks on the feed belt must be physically separated. We can model this as part of the plant, in effect saying that the plant is physically incapable of placing blanks too close to each other. Or we can allow blanks to be put arbitrarily close on the feed belt and then add a specification that forbids the blanks to be placed improperly. In either case, the controlled system would exhibit the same behavior (assuming we got everything modeled correctly), but from two different conceptualizations of the system.

As an example of this plant/specification separation problem, we can note that in [114], the feed belt can initially not move forwards unless a metal blank is put on it. However, in practice the feed belt motor can be started and stopped irrespective of whether any metal blanks are on the feed belt or not. We may not want to move the feed belt unless a metal blank is on it, but this is then a specification and not an inherent property of the plant.

The Plant Model

In the plant model the respective device and configuration components should be clearly distinguishable.

The feed belt consists of a motor and a sensor. The physical local configuration requires the motor to be running for the sensor to be able to sense a metal blank. When a blank has been sensed by the sensor, it cannot be “unsensed” unless the motor continues to run to make the blank leave the sensor. Furthermore, once a blank has been sensed, stopping the motor will make the blank remain at the sensor. After being stopped, the motor can be started again. The feed belt can be described by the EFA in Figure 5.2.

There are three variables associated with the feed belt, fb_motor, fb_sensor and fb_size. The first two represent the state of the motor and sensor, respectively. It can be argued that these variables are superfluous, since the same information can be conveyed by the locations of the EFA in Figure 5.2. But since actions cannot refer to locations currently, these Boolean variables are necessary. The third variable, fb_size models the number of blanks on the feed belt. Thus, its domain is $\{0, 1, 2\}$,

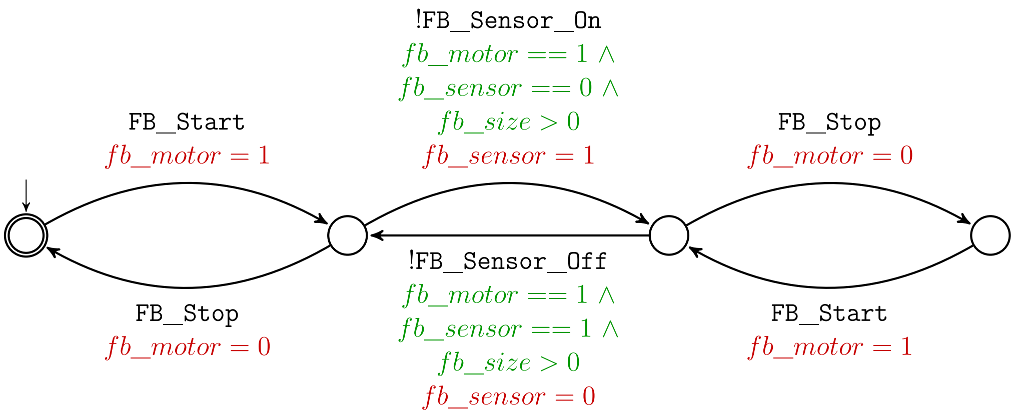


Figure 5.2: Feed belt model.

since we can have at most two blanks on the belt.

Note that the feed belt model does not include any "knowledge" about the crane or the table. This is configuration matter that will be treated shortly, once we have discussed the rotating table.

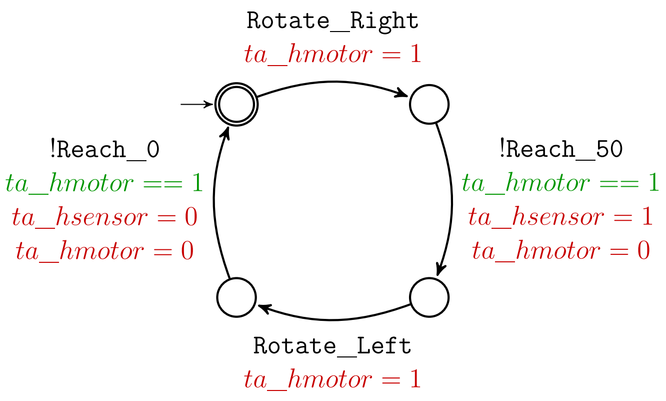
The rotating elevating table can rotate horizontally and elevate vertically. The rotation is constrained to two distinct positions, the 0 degree in line with the feed belt, which is the initial state, and the rotated 50 degrees to align with the robot pick-up position. Likewise, the elevation is constrained to two positions, the initial bottom position aligned with the feed belt and the top position aligned with the robot pick-up position. The EFA modeling the rotation and elevation are depicted in Figure 5.3.

There are five Boolean variables associated with the table, two for the motors, ta_hmotor and ta_vmotor, representing the horizontal and vertical movement motors, respectively; two for the position sensors, $ta\_hsensor$ and $ta\_vsensor$, sensing the horizontal and vertical positions, respectively; and ta_size, which records whether a blank is on the table or not.

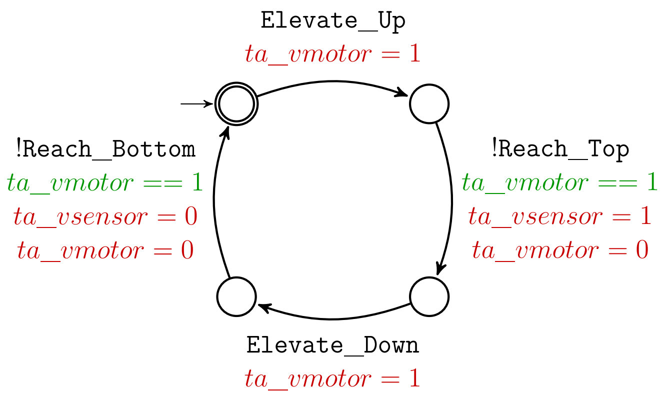
Note that since the up-down and left-right movements are independent, there is no local configuration to model any interaction between the two sub-models of Figure 5.3.

There must, however, exist system configuration to model the passing of a blank from the feed belt to the table. If a blank is at the feed belt sensor, and the feed belt motor runs, the blank will eventually disappear from the feed belt. But where will it go? If the table is in its initial position, down and at 0 degrees, then the blank will pass from the feed belt to the table. If the table is at any other position the blank will fall on the floor. Thus, the configuration that ties together the feed belt and the table, looks like in Figure 5.4.

The configuration encodes the logical effect when the blank leaves the feed belt. If the table is aligned correctly with both horizontal and vertical movement stopped, then the blank is taken from the feed belt to the table. This is captured by the upper self-loop in Figure 5.4. However, if the table is not properly aligned or it still moves, then the blank simply disappears from the feed belt.



(a) Horizontal rotation.



(b) Vertical movement.

Figure 5.3: EFA for rotating elevating table.

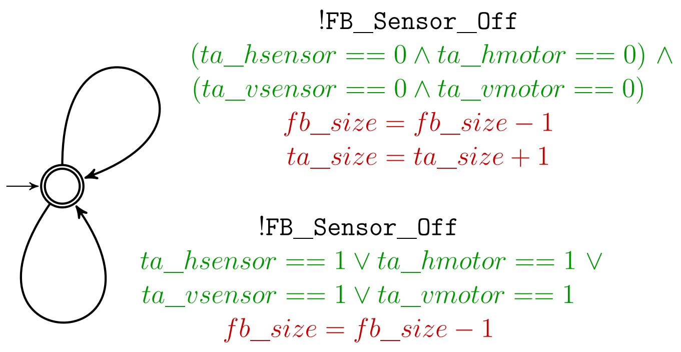

!FB_Sensor_Off
$(ta\_hsensor == 0 \land ta\_hmotor == 0) \land$
$(ta\_vsensor == 0 \land ta\_vmotor == 0)$
$fb\_size = fb\_size - 1$
$ta\_size = ta\_size + 1$
!FB_Sensor_Off
$ta\_hsensor == 1 \lor ta\_hmotor == 1 \lor$
$ta\_vsensor == 1 \lor ta\_vmotor == 1$
$fb\_size = fb\_size - 1$

Figure 5.4: Configuration for feed belt and rotating table.

In some sense, the variable *ta_size* implements a *virtual sensor*. Since there is no actual sensor on the device to tell if a blank is on it or not, this has to be taken care of by the control system through maintaining the internal variable *ta_size* that acts as such a sensor. Also note that the blank falling on the floor is possible but unwanted behavior. Thus, we will have to formulate a specification to avoid it.

The robot consists of several components. It has two arms that can individually be extracted and retracted. Each arm is equipped with an electromagnet that can individually be activated and deactivated. There is also a motor to rotate the robot clockwise and anti-clockwise.

A typical manufacturing robot comes equipped with a sophisticated local control system. Thus, it is reasonable to regard the robot as a rather "intelligent" device, which handles much of its own safety. However, in our case we assume that the robot only handles its own local components and their configuration, it does not interact by itself with other devices. In particular, the robot is not "intelligent" enough to not risk colliding with the press if an arm is extended towards the press while the press is forging a blank. Thus, this unwanted behavior needs to be specified and imposed on the closed-loop system by the supervisor.

Though the robot can rotate arbitrarily, only rotation to three positions are relevant in this particular system, namely rotating so that

1. arm_1 can reach a blank on the rotating table;
2. arm_1 can load the press, and arm_2 can put a forged blank on the deposit belt;
3. arm_2 can unload the press.

So, the rotation of the robot is quantized to these specific positions, and no others will be possible in the system. Admittedly, this is a case of where the plant model imposes restrictions that could equally be regarded as specification, but as the local controller of the robot only allows rotation to these specific positions, this is an inherent behavior of the physical plant. Note that this immediately takes care of the requirement to not rotate the robot too far to any side; it is simply not possible in the plant. The model for the robot $arm2$ is given in Figure 5.5.

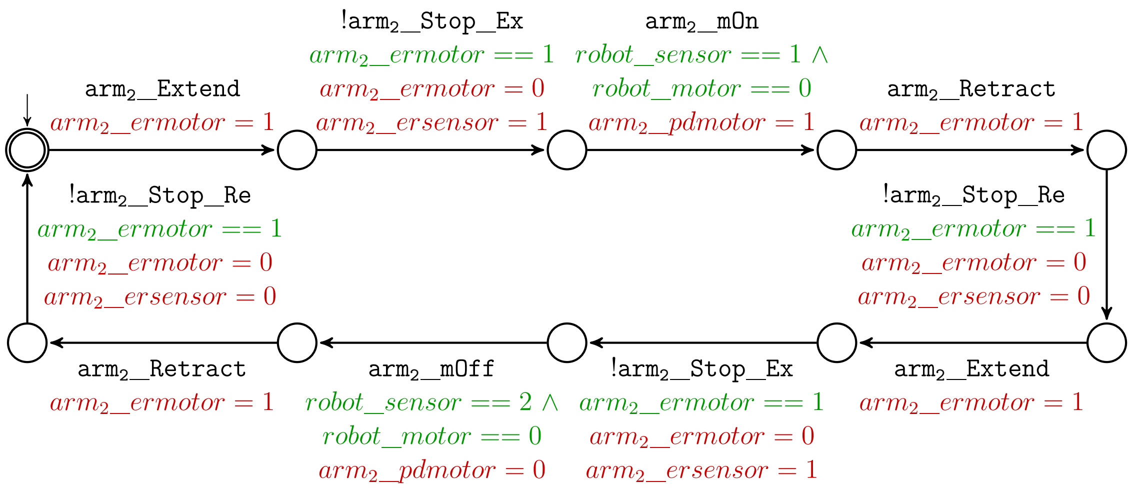


Figure 5.5: EFA for robot arm 2. The model for arm 1 looks the same, except all "arm_2" are replaced by "arm_1".

As can be seen in Figure 5.5, the arm can extend to carry out the task to pick up (event $\texttt{arm}_2\texttt{\_mOn}$) or drop (event $\texttt{arm}_2\texttt{\_mOff}$) a blank. As soon as the respective task is done, the arm retracts back. There are four Boolean variables:

1. $arm_2\_ersensor$, to indicate whether the arm is extended or retracted;
2. $arm_2\_ermotor$, to extend or retract the arm;
3. $arm_2\_pdmotor$, to activate or deactivate the magnet to pick up or drop the blanks;
4. arm_2_loaded, to tell whether there is a blank loaded.

The first three of these variables represent physical signals while arm_2_loaded is a virtual sensor that keeps track of information necessary for control but for which no physical sensor exists. Note that arm_2_loaded does not appear in the EFA for $arm2$, since its value depends on the interaction between the press and the robot; when the magnet of $arm2$ is activated with the arm extended into the press, arm_2_loaded will be set to $true$ only if the press has a blank. This is shown in Figure 5.7.

The local control system of the robot also guarantees that the robot never rotates with any arm extended, see the updates for the events $\texttt{Ro\_Rotate\_Left}$ and $\texttt{Ro\_Rotate\_Right}$ in Figure 5.6. Thus, it is not possible for the robot to collide with any device while rotating. Furthermore, the robot only activates or deactivates its electromagnets when an arm is fully extracted, While retracted, the robot maintains the state of the electromagnet, so it guarantees that no blanks held by it are dropped while rotating.

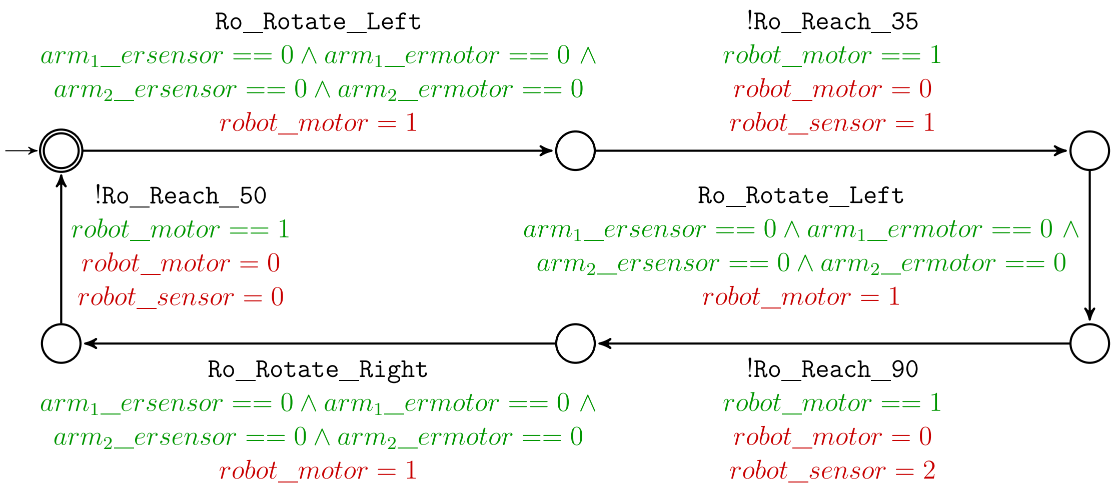
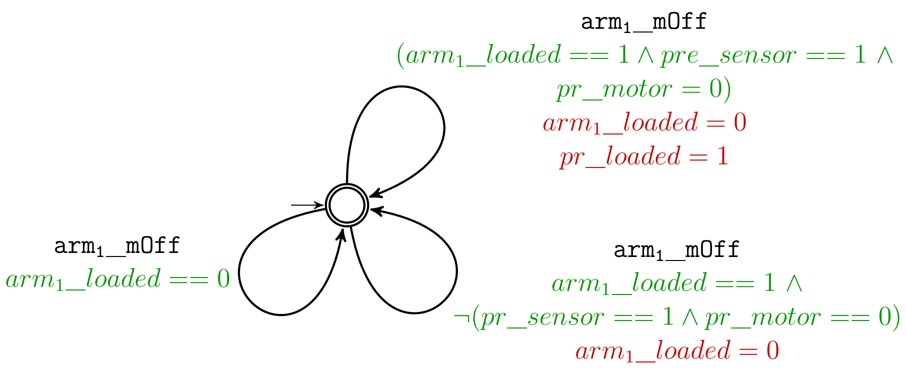


Figure 5.6: EFA for the robot base, which manages the rotation of the robot.

The robot model consists of these independent component models, the robot base and the two arms. However, due to the physical system configuration, these must be tied together with other devices through configurations. The two arms interact



arm₁_mOff
$(arm_1\_loaded == 1 \wedge pre\_sensor == 1 \wedge$
$pr\_motor = 0)$
$arm_1\_loaded = 0$
$pr\_loaded = 1$
arm₁_mOff
$arm_1\_loaded == 0$
arm₁_mOff
$arm_1\_loaded == 1 \wedge$
$\neg(pr\_sensor == 1 \wedge pr\_motor == 0)$
$arm_1\_loaded = 0$

Figure 5.7: Configuration for the interaction between robot $arm1$ and the press.

with the press and the elevating table. The model for the configuration governing the interaction between the press and robot $arm1$ is shown in Figure 5.7.

The configuration for robot and the press in Figure 5.7 models that through the event `arm₁_mOff`, a blank can be loaded on the press by $arm1$ (top self-loop), or the blank can fall on the floor. If $arm1$ does not hold a blank, that is, $arm_1\_loaded == 0$, nothing will happen (left-most self-loop).

The Specification Model

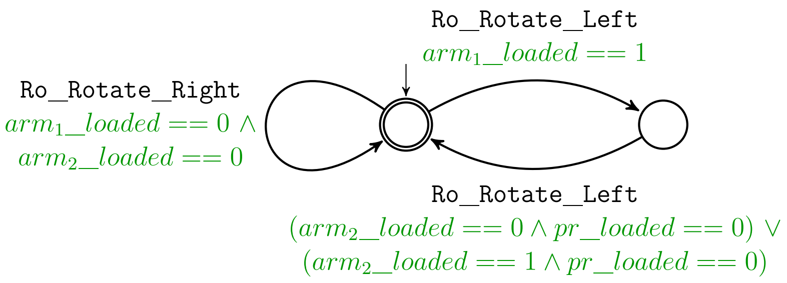
The specification describes the desired or forbidden behavior. Naturally, only behavior that is possible in the plant is relevant, but there is really no reason to require the specification to *only* describe possible behavior. In fact, it can be beneficial to describe more than what is possible by a specific plant. The specification could, for instance, include behavior that is relevant for another more capable plant, so that the specification would be useful also in the future when the current plant is extended. Or it could simply be easier to describe a larger behavior than the possible one as desired (or forbidden). In any case, it is the intersection of the possible behavior as described by the plant, and the desired/forbidden behavior as described by the specification that is of real importance for the synthesis algorithm.

For the robot there are two specifications, one for the rotation Figure 5.8a, and one for the arms Figure 5.8b.

The robot is allowed to turn right only if both arms are unloaded (see the self-loop in Figure 5.8a). This means that it has dropped its blanks. Furthermore, the robot is allowed to turn left (the `Ro_Rotate_Left` event) from the table to the press if $arm1$ is loaded. Then, it can continue to turn left in two situations:

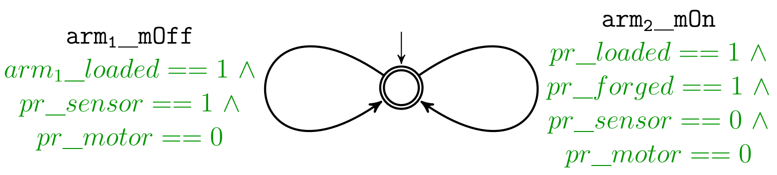
1. For the first time, the press is empty, and $arm2$ is not loaded.

2. If the press is loaded, $arm2$ needs to wait for it to forge the blank and pick the blank up before turning right.



Ro_Rotate_Left
$arm_1\_loaded == 1$
Ro_Rotate_Right
$arm_1\_loaded == 0 \wedge$
$arm_2\_loaded == 0$
Ro_Rotate_Left
$(arm_2\_loaded == 0 \wedge pr\_loaded == 0) \vee$
$(arm_2\_loaded == 1 \wedge pr\_loaded == 0)$

(a) Specification for the robot rotation.



(b) Specification for the robot arms.

Figure 5.8: The two specifications for the robot.

These two conditions are captured by the guard on the `Ro_Rotate_Left` event going from right to left in the bottom of Figure 5.8a.

The arm specification Figure 5.8b, allows $arm1$ to drop a blank (event $\texttt{arm}_1\texttt{\_mOff}$) and $arm2$ to pick a blank (event $\texttt{arm}_2\texttt{\_mOn}$) only under the correct circumstances. For $arm1$ this means that the arm must be loaded ($arm1\_loaded == 1$), the press is in the middle position ($pr\_sensor == 1$) and is not moving ($pr\_motor == 0$). Note that the robot position does not need to appear in the specification, since the local controller of the robot takes care of only deactivating the magnet in the right position, compare Figure 5.5.

Similarly, the specification for $arm2$ allows $arm2$ to pick a blank (event $\texttt{arm}_2\texttt{\_mOn}$), only if there is a forged blank in the press, the press is in the bottom position and it is not moving.

5.3 Synthesis and Guard Extraction

Given a set of EFA that describe the plant and the specification, synthesis of a supervisor that dynamically restricts the plant behavior to remain within the specification, is the next step. In this section, we discuss the results obtained by applying the proposed symbolic algorithms on the EFA modeling the manufacturing system. In particular, we first apply the extended workset algorithm of [Paper 2] on the considered EFA to compute the BDD representing all the states belonging to the supervisor. Subsequently, making use of the set of constructed partial transition relations, we apply the symbolical algorithm of [Paper 1] to generate the corresponding guards.

Table 5.1: Comparison between two symbolic synthesis approaches.

		The approach of [38]		The approach of [Paper 2]	
Reachable	Supervisor	BDD Peak	Computation Time	BDD Peak	Computation Time
3.61×10^7	1.06×10^7	34689	40 sec.	1658	20 sec.

Table 5.2: Events and the numbers of logical terms of their supervisor guards.

Event	Number of terms	Event	Number of terms
`Elevate__Up`	55	`Ro__Rotate__Right`	285
`arm`$_1$`__Extend`	154	`Crane__mOff`	173
`Add__Blank`	768	`FB__Start`	514
`Remove__Blank`	24	`Crane__mOn`	42
`Ro__Rotate__Right`	115	`arm`$_2$`__Extend`	14

For the EFA model of the manufacturing system, the number of reachable states of the composed system is 3.61×10^7. Among the reachable states, the number of states belonging to the synthesized supervisor is 1.06×10^7. Hence, explicit enumeration of such an enormous is hardly tractable. As shown in Table 5.1, by applying the symbolic approach presented in [Paper 2] that uses the workset algorithm to explore the state-space, the supervisor can be synthesized in less than 20 seconds. The maximal number of BDD nodes, i.e., BDD Peak, as the indication of the maximal memory usage during the algorithm execution, is 1658. As a comparison, the symbolic algorithm introduced in [38] is applied on the same EFA model and the results are reported in Table 5.1. The supervisor is synthesized within 40 seconds. The maximal number of BDD nodes allocated by the symbolic algorithm of [38] is 34689. Hence, for the manufacturing system, the algorithm of [Paper 2] outperforms the algorithm of [38] in both of the time and memory requirements.

As mentioned earlier, though the BDD-based synthesis is able to treat systems of this size, it also brings a problem in that the result is not straightforwardly accessible. Since the original EFA have been reformulated and encoded, it is cumbersome for human users to relate each state to the corresponding BDD variables. To this end, it would be much more convenient and natural to represent the BDD-based supervisor in a form similar to the originally given models. This can be done by applying the symbolic approach represented in [Paper 1] and [70] on the symbolically represented supervisor to extract and represent the supervisor as Boolean expressions.

There are in total 52 events in the EFA modeling the considered system. Among these, 42 events do not get assigned by any supervisor guards; these events are always enabled by the supervisor. There are merely 10 events that do get supervisor guards assigned, the guard sizes are given in Table 5.2. Note that the shape and size of the supervisor guards vary significantly with the variable ordering of the BDD. Possibly, alternative variable ordering could give smaller sized guards than the ones in Table 5.2.

We will exemplify with the event `arm`$_2$`_Extend` that represents extending the robot arm $arm2$ for unloading the press, see Figure 5.5. One conjunctive part[1] of the generated guard for this event is:

$$(pr\_loaded \neq 0) \ \wedge \ \Big((arm_1\_loaded \neq 0) \ \vee \ (arm_2.curr \neq arm_2\_initial)\Big). \quad (5.1)$$

When the expression (5.1) is true, the event `arm`$_2$`_Extend` is enabled by the supervisor. More specifically, the event `arm`$_2$`_Extend` is enabled when the press is loaded, and arm_1 is loaded or arm_2 is currently not in its initial location. Thus, there are two situations.

1. The press is loaded and arm_1 is loaded. This describes a situation where arm_1 loads a blank from the table and the robot starts to turn left to the press. The press is loaded, which indicates that the blank in it needs to be forged and unloaded by arm_2. Under this circumstance, arm_2 can extend if the robot reaches the press and the press stays at the bottom.

2. The press is loaded and the current location of arm_2 is not in the initial location (i.e., $arm_2\_initial$). From the arm_2 plant model (see Figure 5.5), it can be observed that the event `arm`$_2$`_Extend` is enabled from only two locations; the initial location and the location where arm_2 has retracted. Therefore, the current location of arm_2 is in the latter location. Thus, the guard describes the situation where arm_1 has loaded a blank in the press while arm_2 is positioned above the deposit belt waiting to extend and unload the forged blank.

In this section, we apply the proposed symbolic algorithms on the EFA model of the manufacturing system obtained from Section 5.2. The reported results have clearly revealed the superiority of the extended workset algorithm of [Paper 2] with respect to the prior work [38]. The synthesized supervisor, together with the constructed partial transition relations, is then fed into the algorithm of [Paper 1] to extract additional guards to represent the supervisor, which makes the implementation of the supervisor straightforward in an industrial control system. Nevertheless, for large systems, the supervisor guards typically become unwieldy due to their large numbers of terms. One reason is that it is not easy to predict before hand how the variable ordering affects the number of terms for the supervisor guards. Still, using the don't-care states together with some heuristic rules, some kind of compactness can be achieved.

5.4 Summary

In this chapter, we have shown by a case study how: (i) EFA-based modeling can lead to a compact and rather comprehensible representation of a large system; (ii) BDD-

[1]The full representation of the genrated guard for the event `arm`$_2$`_Extend` is described in [113]. For simplicity and understandability, we only exemplify the considered part in this chapter.

based synthesis that are presented in the appended papers can efficiently compute a supervisor for such a large system of practical complexity and size; (iii) guards can represent that supervisor as Boolean conditions that directly relate to the originally given plant.

In [Paper 5], another case study is presented to demonstrate the applicability of the proposed approaches of [Paper 3] and [Paper 4] to the elimination of deadlocks in multithreaded software, which has been previously addressed by the Gadara project [116, 22, 17]. By modeling a special kind of Petri net (i.e., Gadara net [22]) that models the primitive lock acquisition into the equivalent EFA, [Paper 5] demonstrates in detail how the set of minimal boundary unsafe states are symbolically computed. In addition, the guard generation procedure is further extended to generate guards from the BDD representing the minimal boundary unsafe states. For a more detailed explanation of this case study, refer to [Paper 5].

Chapter 6

Contributions

Part II of this thesis consists of five papers. In this chapter, the contributions of the included papers are summarized.

Paper 1

> Zhennan Fei, Sajed Miremadi, Knut Åkesson, Bengt Lennartson. *Symbolic State-Space Exploration and Guard Generation in Supervisory Control Theory.* Agents and Artificial Intelligence – Communications in Computer and Information Science, by Joaquim Filipe and Ana Fred (eds), Springer, vol. 271, pp. 161–175, 2013.

The main contribution of [Paper 1] is the adaption of a symbolic supervisory synthesis approach from the prior work [34] to the guard generation procedure [70], making it more applicable for industrial applications. In particular, by using one of the partitioning techniques, i.e., the disjunctive partitioning technique, the proposed approach splits the monolithic transition relation of DES modeled as deterministic FA into a set of partial transition relations, that are used iteratively for the state-space exploration. The guard generation procedure is tailored to use the partitioned structure to extract the simplified guards and attach them to the original models. Furthermore, a comparison of algorithm efficiency between the two partitioning techniques is made by applying them to a set of benchmark examples.

Paper 2

> Zhennan Fei, Sajed Miremadi, Knut Åkesson, Bengt Lennartson. *Efficient Symbolic Supervisor Synthesis for Extended Finite Automata.* IEEE Transactions on Control Systems Technology, in press, 2014.

In [Paper 1], the modeling formalism that is employed to model DES is deterministic FA. In [Paper 2], we focus on the EFA modeling framework and the corresponding symbolic supervisor synthesis. The contributions of [Paper 2] include: (i) Applying the disjunctive partitioning technique to DES modeled as EFA by suggesting a new way to construct the set of partial transition relations for the considered EFA model. Each such partial transition relation corresponds to a particular event. (ii) Proposing a new algorithm that exploits the disjunctive partial transition relations to compute a BDD representing reachable states. It is shown through solving a set of benchmark supervisory control problems that the proposed algorithm has improved scalability in comparison to the symbolic approach presented in [38] due to its ability to explore the state-space in a structured way, which can significantly alleviate the problem with large intermediate BDDs.

Paper 3

Zhennan Fei, Spyros Reveliotis, Sajed Miremadi, Knut Åkesson. *A BDD-Based Approach for Designing Maximally Permissive Deadlock Avoidance Policies for Complex Resource Allocation Systems.* Submitted for a possible journal publication, 2014.

Starting from [Paper 3], we focus on the deadlock avoidance for resource allocation systems that is a particular DES application domain arising in many contemporary technological systems. The main contribution of [Paper 3] is the proposed symbolic framework for the efficient development of the maximally permissive DAP of the considered RAS. In particular, we first show how the considered RAS can be recast into the corresponding EFA model without losing any information necessary for solving the deadlock avoidance problem. Secondly, we present a series of symbolic algorithms for computing the minimal boundary unsafe states from the BDD-based representation of the underlying state-space. The experimental results reveal that the proposed symbolic computation enables the deployment of the maximally permissive DAP for complex RAS with very large structure and state-spaces, with limited time and memory requirements.

Paper 4

Zhennan Fei, Spyros Reveliotis, Knut Åkesson. *Symbolic Computation of Boundary Unsafe States in Complex Resource Allocation Systems using Partitioning Techniques.* Submitted for a possible journal publication, 2014.

The work presented in [Paper 4] enhances the results of [Paper 3] by introducing an attractive algorithm for computing the boundary unsafe states in complex RAS. Instead of performing all the computations on a monolithic BDD representing the RAS state-space, this algorithm performs the identification of unsafe states individually on the event-based disjunctive representation of the transition relation of the considered RAS-modeling EFA, i.e., the set of (substantially) smaller BDDs with each one corresponding to a particular event. In comparison with the presented approach in [Paper 3], the algorithm presented in this work has better scalability due to the lower memory requirement for its BDD-based implementation. Furthermore, with a series of minor modifications, the proposed algorithm can be parallelized to further improve its time requirement. The algorithm can be also easily extended to account for uncontrollable RAS dynamics, where uncontrollability is defined either in terms of the timing of some process-advancing events, or in terms of the routing decisions at the various processing stages.

Paper 5

Zhennan Fei, Knut Åkesson, Spyros Reveliotis. *Symbolic Computation and Representation of Deadlock Avoidance Policies for Complex Resource Allocation Systems with Application to Multithreaded Software.* Submitted to the 53rd IEEE Conference on Decision and Control (CDC), 2014.

The work presented in [Paper 5] extends the symbolic framework that was described in [Paper 3] and [Paper 4] by introducing a procedure that generates a set of comprehensible "guard" predicates to represent the resulting DAP. By attaching them to the original model, the generated predicates guard transitions to the states that dominate some elements in the set of minimal boundary unsafe states. Furthermore, this work applies the developed approaches to the problem of deadlock avoidance in shared-memory multithreaded software, which has been previously addressed by the Gadara project [116, 22, 17].

Chapter 7

Conclusions and Future Work

As one of the main obstacles when it comes to the analysis of large-scale DES, the state-space explosion problem has been well-studied for decades in the relevant research communities. In brief, the problem arises from the inherent combinatorial growth of the number of states as monolithic models are built. Thus, explicitly enumerating synchronized models with huge state-space fails due to time and memory limits. On the other hand, a well-known approach for combating the state-space explosion is to symbolically represent discrete event system models and compute supervisors by using BDD, a compact and operation-efficient data structure for representing Boolean functions.

One objective of this thesis is to develop symbolic BDD-based algorithms for the effective and computationally efficient development of the maximally permissive control logic to guarantee that the behaviors of systems fulfill the specifications. Both the considered plant and specification are modeled in the automaton-based modeling framework, either as a set of deterministic FA or as a set of EFA. To perform the symbolic exploration on the state-space of the composed system efficiently, two strategies are employed: (i) Making use of the disjunctive partitioning technique to construct a set of partial transition relations with simpler structure and smaller sizes to represent the monolithic transition relation of the composed system. Based on the disjunctive representation, a series of algorithms are proposed to traverse the state-space by performing the symbolic computation on these partial transition relations iteratively. (ii) Taking advantage of the structural properties possessed by the considered systems, e.g., RAS, the target control policy can be obtained by performing a partial exploration of the underlying state-space. As presented in the included papers these two strategies can be combined to result in a more efficient symbolic computation for various classes of resource allocation systems.

The second objective of the thesis is to represent the result obtained from the symbolic approaches in a more comprehensible and transparent manner. As mentioned earlier, representing the synthesized control logic that usually consists of a large number of states is a challengeable task. To this end, the thesis provides two extensions to the previously published procedure for extracting guards and attaching them to the original model to represent the resulting supervisor. First, by tailoring the guard procedure to use the disjunctive representation of the state-space to compute the sets of states that are necessary for the guard generation, the guard generation procedure is made more applicable for industrially interesting applications. Secondly, the procedure is extended to generate comprehensible guards from the maximally permissive DAP for RAS. By attaching them to the original RAS-modeling EFA, the generated predicates guard transitions to the states that dominate some elements in the set of minimal boundary unsafe states.

The proposed framework including a number of BDD-based algorithms that are presented in the thesis has been implemented in the supervisory control tool Supremica and applied to a set of academic and industrial DES for the supervisor synthesis and guard generation. Also, extensive benchmarks reveal that the presented methodology can handle complex RAS structures with large state spaces that were previously intractable. Hence, the presented framework holds a strong potential for providing robust, practical and efficient solutions to a broad range of deadlock avoidance and liveness-enforcing supervision problems that are experienced in the considered DES application domain.

Future work

There are a number of directions towards which we could improve and extend the work in future.

The main focus of this thesis has been on the symbolic supervisor synthesis for DES modeled as deterministic FA or EFA. It would be interesting to investigate the possibility of combining our symbolic approach with the techniques developed from hierarchical approaches such as STS [65], or the compositional approaches [54] and [56], to further improve the scalability and performance of our BDD-based symbolic algorithms for supervisor synthesis. As mentioned in Chapter 4, the variable ordering has a key impact on the sizes of BDDs in symbolic computation but finding an optimal variable ordering is NP-complete. In this thesis, all the presented symbolic algorithms employ the heuristics developed in [63]. It is believed that more investigations are worth performing towards finding a better sub-optimal variable ordering, in order to handle larger and more complicated supervisory control problems.

In the area of deadlock avoidance for RAS, it is also interesting to investigate the possibility of extending the applicability of the presented symbolic algorithms to RAS with infinite state spaces, for instance RAS with reader/writer (R/W) locks that is considered in [117]. Finally, to be able to further shorten the guards, it is also interesting to investigate the possibility of designing advanced techniques and heuristics by exploiting the special structure of RAS.

References

[1] M. Dowson, “The Ariane 5 software failure,” *ACM SIGSOFT Software Engineering Notes*, vol. 22, no. 2, p. 84, 1997.

[2] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Springer, 2008.

[3] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking.* MIT Press, 1999.

[4] C. Baier and J.-P. Katoen, *Principles of Model Checking.* The MIT Press, 2008.

[5] P. Ramadge and W. Wonham, “Supervisory control of a class of discrete event processes,” *SIAM Journal of Control and Optimization*, vol. 25, no. 1, pp. 635–650, 1987.

[6] ——, “The control of discrete event systems,” *Proceedings of the IEEE, Special Issue on Discrete Event Dynamic Systems*, vol. 77, no. 1, pp. 81–98, 1989.

[7] T. Murata, “Petri nets: Properties, analysis and applications,” *Proceedings of the IEEE*, vol. 77, pp. 541–580, 1989.

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

[9] S. A. Reveliotis, *Real-time Management of Resource Allocation Systems: A Discrete Event Systems Approach.* NY, NY: Springer, 2005.

[10] J. Ezpeleta, J. M. Colom, and J. Martinez, “A Petri net based deadlock prevention policy for flexible manufacturing systems,” *IEEE Trans. on R&A*, vol. 11, pp. 173–184, 1995.

[11] S. A. Reveliotis and P. M. Ferreira, “Deadlock avoidance policies for automated manufacturing cells,” *IEEE Trans. on Robotics & Automation*, vol. 12, pp. 845–857, 1996.

[12] M. P. Fanti, B. Maione, S. Mascolo, and B. Turchiano, “Event-based feedback control for deadlock avoidance in flexible production systems,” *IEEE Trans. on Robotics and Automation*, vol. 13, pp. 347–363, 1997.

[13] S. A. Reveliotis, “Conflict resolution in AGV systems,” *IIE Trans.*, vol. 32(7), pp. 647–659, 2000.

[14] N. Wu and M. Zhou, “Resource-oriented Petri nets in deadlock avoidance of AGV systems,” in *Proceedings of the ICRA'01.* IEEE, 2001, pp. 64–69.

[15] S. Reveliotis and E. Roszkowska, “Conflict resolution in free-ranging multi-vehicle systems: A resource allocation paradigm,” *IEEE Trans. on Robotics*, vol. 27, pp. 283–296, 2011.

[16] A. Giua, M. P. Fanti, and C. Seatzu, “Monitor design for colored Petri nets: an application to deadlock prevention in railway networks,” *Control Engineering Practice*, vol. 10, pp. 1231–1247, 2006.

[17] H. Liao, Y. Wang, H. K. Cho, J. Stanley, T. Kelly, S. Lafortune, S. Mahlke, and S. Reveliotis, “Concurrency bugs in multithreaded software: Modeling and analysis using Petri nets,” *Discrete Event Systems: Theory and Applications*, vol. 23, pp. 157–195, 2013.

[18] H. Liao, Y. Wang, J. Stanley, S. Lafortune, S. Reveliotis, T. Kelly, and S. Mahlke, “Eliminating concurrency bugs in multithreaded software: A new approach based on discrete-event control,” *IEEE Trans. on Control Systems Technology*, vol. 21, no. 6, pp. 2067–2082, 2013.

[19] P. Gohari and W. Wonham, “On the complexity of supervisory control design in the RW framework,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 30, no. 5, pp. 643–652, 2000.

[20] S. B. Akers, “Binary Decision Diagrams,” *IEEE Transactions on Computers*, vol. 27, pp. 509–516, 1978.

[21] R. E. Bryant, “Symbolic Boolean manipulation with ordered binary-decision diagrams,” *ACM Comput. Surv.*, vol. 24, no. 3, pp. 293–318, 1992.

[22] Y. Wang, H. Liao, S. Reveliotis, T. Kelly, S. Mahlke, and S. Lafortune, “Modeling and analysis of a special class of Petri nets arising in multithreaded programs,” in *CDC 2009*, 2009.

[23] K. Åkesson, M. Fabian, H. Flordal, and A. Vahidi, “Supremica—A tool for verification and synthesis of discrete event supervisors,” in *11th Mediterranean Conference on Control and Automation*, 2003.

[24] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, “Supremica - an integrated environment for verification, synthesis and simulation of discrete event systems,” in *the 8th International Workshop on Discrete Event Systems*, 2006, pp. 384–385.

[25] Supremica, “`www.supremica.org`. The official website for the Supremica project,” 2014.

[26] T. Murata, “Petri nets: Properties, analysis and applications,” *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[27] K. Inan and P. Varaiya, “Algebras of discrete event models,” *Proceedings of the IEEE*, vol. 77, no. 1, pp. 24–38, 1989.

[28] J. C. M. Baeten, D. A. Van Beek, B. Luttik, J. Markovski, and J. Rooda, “A process-theoretic approach to supervisory control theory,” in *American Control Conference*, 2011, pp. 4496–4501.

[29] W. Wonham, *Supervisory Control of Discrete Event Systems*, Toronto, Canada, 2011.

[30] C. A. R. Hoare, *Communicating sequential processes*, ser. Series in Computer Science. ACM, 1978, vol. 21, no. 8.

[31] B. Lennartson, F. Basile, S. Miremadi, Z. Fei, M. Hosseini, M. Fabian, and K. Akesson, “Supervisory control for state-vector transition models – a unified approach,” *Automation Science and Engineering, IEEE Transactions on*, vol. 11, no. 1, pp. 33–47, 2014.

[32] W. M. Wonham and P. Ramadge, “On the Supremal Controllable Sublanguage of a Given Language,” *SIAM Control and Optimization*, vol. 25, no. 3, pp. 637–659, 1987.

[33] R. Kumar, V. K. Garg, and S. I. Marcus, “On Controllability and Normality of DEDS,” *Systems and Control Letters*, vol. 17, pp. 157–168, 1991.

[34] A. Vahidi, M. Fabian, and B. Lennartson, “Efficient supervisory synthesis of large systems,” *Control Engineering Practice*, vol. 14, no. 10, pp. 1157–1167, Oct. 2006.

[35] K. Åkesson, “Methods and tools in supervisory control theory: operator aspects, computation efficiency and applications,” Ph.D. dissertation, Signals and Systems,Chalmers University of Technology, Göteborg, Sweden, 2002.

[36] L. Ouedraogo, R. Kumar, R. Malik, and K. Akesson, “Nonblocking and Safe Control of Discrete-Event Systems Modeled as Extended Finite Automata,” *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 3, pp. 560–569, 2011.

[37] M. Sköldstam, K. Åkesson, and M. Fabian, “Supervisory Control Applied to Automata Extended with Variables - Revised,” Chalmers University of Technology, Tech. Rep., 2008.

[38] S. Miremadi, B. Lennartson, and K. Åkesson, “A BDD-based approach for modeling plant and supervisor by extended finite automata,” *IEEE Transactions on Control Systems Technology*, vol. 20, no. 6, pp. 1421–1435, 2012.

[39] W. Wonham and P. Ramadge, “Modular Supervisory Control of Discrete-Event Systems,” *Mathematics of Control Signals and Systems*, vol. 1, no. 1, pp. 13–30, 1988.

[40] K. Rudie and W. Wonham, “Think globally, act locally: decentralized supervisory control,” *IEEE Transactions on Automatic Control*, vol. 37, no. 6, pp. 1692–1708, 1992.

[41] K. Wong and W. Wonham, “Modular control and coordination of discrete-event systems,” *Discrete Event Dynamic Systems*, vol. 8, no. 3, pp. 247–297, 1998.

[42] B. A. Brandin, R. Malik, and P. Dietrich, “Incremental system verification and synthesis of minimally restrictive behaviors,” in *American Control Conference*, 2000, pp. 4056–4061.

[43] M. de Queiroz and J. Cury, “Modular control of composed systems,” in *Proceedings of the 2000 American Control Conference*, vol. 6, no. June, 2000, pp. 4051–4055.

[44] K. Åkesson, H. Flordal, and M. Fabian, “Exploiting Modularity for Synthesis and Verification of Supervisors,” in *15th IFAC World Congress*, 2002.

[45] 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*, vol. 20, no. 1, pp. 139–185, 2010.

[46] H. Zhong and W. Wonham, “On Consistency of Hierarchical Supervision in Discrete-Event Systems,” *IEEE Transactions on Automatic Control*, vol. 35, no. 10, pp. 1125–1134, 1990.

[47] Y. Brave, “Control of discrete event systems modeled as hierarchical state machines,” *Automatic Control, IEEE Transactions on*, vol. 38, no. 12, pp. 1803–1819, 1993.

[48] R. J. Leduc, “Hierarchical interface-based supervisory control,” Ph.D. dissertation, Electrical and Computer Engineering,Toronto, Toronto, Canada, 2002.

[49] R. Song and R. J. Leduc, “Symbolic Synthesis and Verification of Hierarchical Interface-based Supervisory Control,” in *8th Discrete Event Systems, WODES ’06*, Ann Arbor, MI, USA, 2006, pp. 419–426.

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

[51] R. C. Hill, J. E. R. Cury, M. H. de Queiroz, D. M. Tilbury, and S. Lafortune, "Multi-level hierarchical interface-based supervisory control," *Automatica*, vol. 46, no. 7, pp. 1152–1164, 2010.

[52] S. Graf and B. Steffen, "Compositional Minimization of Finite State Systems," in *Proceedings of the 2nd International Workshop on Computer Aided Verification.* Springer, 1991, pp. 186–196.

[53] H. Flordal, "Compositional Approaches in Supervisory Control—with Application to Automatic Generation of Robot Interlocking Policies," Ph.D. dissertation, Signals and Systems,Chalmers University of Technology, Göteborg, Sweden, 2006.

[54] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, "Compositional Synthesis of Maximally Permissive Supervisors Using Supervision Equivalence," *Discrete Event Dynamic Systems*, vol. 17, no. 4, pp. 475–504, 2007.

[55] M. Teixeira, R. Malik, J. Cury, and M. de Queiroz, "Variable abstraction and approximations in supervisory control synthesis," in *American Control Conference (ACC), 2013*, June 2013, pp. 132–137.

[56] S. Mohajerani, R. Malik, and M. Fabian, "A framework for compositional synthesis of modular nonblocking supervisors," *Automatic Control, IEEE Transactions on*, vol. 59, no. 1, pp. 150–162, Jan 2014.

[57] G. Hoffmann and H. Wong-Toi, "Symbolic synthesis of supervisory controllers," in *American Control Conference*, 1992, pp. 2789–2793.

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

[59] G. Hoffmann and H. Wong-Toi, "Symbolic supervisor synthesis for the animal maze," in *Discrete Event Systems: Modeling and Control*, ser. Progress in Systems and Control Theory, S. Balemi, P. Kozák, and R. Smedinga, Eds. Birkhäuser Basel, 1993, vol. 13, pp. 189–197.

[60] E. Asarin, O. Maler, and A. Pnueli, "Symbolic controller synthesis for discrete and timed systems," in *Hybrid Systems II*, ser. Lecture Notes in Computer Science, P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, Eds. Springer Berlin Heidelberg, 1995, vol. 999, pp. 1–20.

[61] J. Gunnarsson, “Symbolic methods and tools for discrete event dynamic systems,” Ph.D. dissertation, Electrical Engineering, Linköping University, Linköping, Sweden, 1997.

[62] ——, “Symbolic tools for verification of large scale deds,” in *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, vol. 1, 1998, pp. 722–727.

[63] A. Vahidi, “Efficient analysis of discrete event systems,” Ph.D. dissertation, Chalmers University of Technology, 2004.

[64] C. Ma, “Nonblocking supervisory control of state tree structures,” Ph.D. dissertation, University of Toronto, 2005.

[65] C. Ma and W. Wonham, “Nonblocking supervisory control of state tree structures,” *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 782–793, 2006.

[66] M. Byröd, B. Lennartson, A. Vahidi, and K. Åkesson, “Efficient Reachability analysis on Modular Discrete-Event Systems using Binary Decision Diagrams,” in *Proceedings of the 8th international Workshop on Discrete Event Systems, WODES’06*, 2006, pp. 288–293.

[67] B. J.R., C. D, and D. E. Long, “Symbolic Model Cheking with Partitioned Transition Relations,” in *A. Halaas and P.B. Denyer, editors, International Conference on Very Large Scale Integration*, Aug. 1991, pp. 49–58.

[68] J. R. Burch, E. M. Clarke, D. E. Long, K. L. Mcmillan, and D. L. Dill, “Symbolic Model Checking for Sequential Circuit Verification,” *IEEE Transactions on ComputerAided Design of Integrated Circuits and Systems*, vol. 13, no. 4, pp. 401–424, 1994.

[69] Z. Fei, K. Åkesson, and B. Lennartson, “Symbolic reachability computation using the disjunctive partitioning technique in supervisory control theory,” in *IEEE International Conference on Robotics and Automation*, 2011, pp. 4364–4369.

[70] S. Miremadi, K. Åkesson, and B. Lennartson, “Symbolic Computation of Reduced Guards in Supervisory Control,” *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 4, pp. 754–765, 2011.

[71] X. Lin and W. Wonham, “On observability of discrete-event systems,” *Information Sciences*, vol. 44, pp. 179–198, 1988.

[72] S. L. Ricker and K. Rudie, “Know Means No: Incorporating Knowledge into Discrete-Event Control Systems,” *IEEE Transactions on Automatic Control*, vol. 45, no. 9, pp. 1656–1668, 2000.

[73] J. Ostroff and W. Wonham, “A framework for real-time discrete event control,” *Automatic Control, IEEE Transactions on*, vol. 35, no. 4, pp. 386–397, Apr 1990.

[74] B. A. Brandin and W. Wonham, “Supervisory Control of Timed Discrete-Event Systems,” *IEEE Transactions on Automatic Control*, vol. 39, no. 2, pp. 329–342, 1994.

[75] B. A. Brandin, “The Modeling and Supervisory Control of Timed DES,” in *Proceedings of the 4th International Workshop of Discrete Event Systems, WODES’98*, 1998, pp. 8–14.

[76] A. Saadatpoor, “Timed State Tree Structures: Supervisory Control and Fault Diagnosis,” Ph.D. dissertation, University of Toronto, 2009.

[77] S. Miremadi, Z. Fei, K. Akesson, and B. Lennartson, “Symbolic representation and computation of timed discrete-event systems,” *Automation Science and Engineering, IEEE Transactions on*, vol. 11, no. 1, pp. 6–19, Jan 2014.

[78] ——, “Efficient Symbolic Supervisory Synthesis for Extended Finite Automata,” *IEEE Transactions on Control Systems Technology (conditionally accepted)*, 2014.

[79] A. Nazeem and S. Reveliotis, “A practical approach for maximally permissive liveness-enforcing supervision of complex resource allocation systems,” *IEEE Trans. on Automation Science and Engineering*, vol. 8, pp. 766–779, 2011.

[80] S. Reveliotis and A. Nazeem, “Deadlock avoidance policies for automated manufacturing systems using finite state automata,” in *Formal Methods in Manufacturing*, J. Campos, C. Seatzu, and X. Xie, Eds. CRC Press / Taylor and Francis, 2014, pp. 169–195.

[81] T. Araki, Y. Sugiyama, and T. Kasami, “Complexity of the deadlock avoidance problem,” in *2nd IBM Symp. on Mathematical Foundations of Computer Science*, 1977, pp. 229–257.

[82] E. M. Gold, “Deadlock prediction: Easy and difficult cases,” *SIAM Journal of Computing*, vol. 7, pp. 320–336, 1978.

[83] M. A. Lawley and S. A. Reveliotis, “Deadlock avoidance for sequential resource allocation systems: hard and easy cases,” *Intl. Jrnl of FMS*, vol. 13, pp. 385–404, 2001.

[84] S. A. Reveliotis, M. A. Lawley, and P. M. Ferreira, “Polynomial complexity deadlock avoidance policies for sequential resource allocation systems,” *IEEE Trans. on Automatic Control*, vol. 42, pp. 1344–1357, 1997.

[85] K. Y. Xing, B. S. Hu, and H. X. Chen, “Deadlock avoidance policy for Petri net modeling of flexible manufacturing systems with shared resources,” *IEEE Trans. on Aut. Control*, vol. 41, pp. 289–295, 1996.

[86] Z. A. Banaszak and B. H. Krogh, “Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows,” *IEEE Trans. on Robotics and Automation*, vol. 6, pp. 724–734, 1990.

[87] F. S. Hsieh and S. C. Chang, “Dispatching-driven deadlock avoidance controller synthesis for flexible manufacturing systems,” *IEEE Trans. on Robotics and Automation*, vol. 10, pp. 196–209, 1994.

[88] M. Lawley, S. Reveliotis, and P. Ferreira, “The application and evaluation of Banker’s algorithm for deadlock-free buffer space allocation in flexible manufacturing systems,” *Intl. Jrnl. of Flexible Manufacturing Systems*, vol. 10, pp. 73–100, 1998.

[89] ——, “A correct and scalable deadlock avoidance policy for flexible manufacturing systems,” *IEEE Trans. on Robotics & Automation*, vol. 14, pp. 796–809, 1998.

[90] J. Park and S. A. Reveliotis, “Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings,” *IEEE Trans. on Automatic Control*, vol. 46, pp. 1572–1583, 2001.

[91] J. Park, S. Reveliotis, M. Lawley, and P. Ferreira, “Correction on the run dap for conjunctive ras presented in "polynomial comlpexity deadlock avoidance policies for sequential resource allocation systems",” *IEEE Trans. on Automatic Control*, vol. 46, p. 672, 2001.

[92] M. Jeng, X. Xie, and M. Y. Peng, “Process nets with resources for manufacturing modeling and their analysis,” *IEEE Trans. on Robotics & Automation*, vol. 18, pp. 875–889, 2002.

[93] Z. Li and M. Zhou, “Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems,” *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 34, no. 1, pp. 38–51, 2004.

[94] F. Tricas, F. Garcia-Valles, J. M. Colom, and J. Ezpeleta, “A Petri net structure-based deadlock prevention solution for sequential resource allocation systems,” in *Proceedings of the ICRA 2005.* IEEE, 2005, pp. 271–277.

[95] M. Zhou and M. P. Fanti (editors), *Deadlock Resolution in Computer-Integrated Systems.* Singapore: Marcel Dekker, Inc., 2004.

[96] Z. Li, M. Zhou, and N. Wu, “A survey and comparison of Petri net-based deadlock prevention policies for flexible manufacturing systems,” *IEEE Trans. Systems, Man and Cybernetics – Part C: Applications and Reviews*, vol. 38, pp. 173–188, 2008.

[97] E. Badouel and P. Darondeau, “Theory of regions,” in *LNCS 1491 – Advances in Petri Nets: Basic Models*, W. Reisig and G. Rozenberg, Eds. Springer-Verlag, 1998, pp. 529–586.

[98] M. Uzam, “An optimal deadlock prevention policy for flexible manufacturing systems using Petri net models with resources and the theory of regions,” *Intl. Jrnl of Advanced Manufacturing Technology*, vol. 19, pp. 192–208, 2002.

[99] A. Ghaffari, N. Rezg, and X. Xie, “Design of a live and maximally permissive Petri net controller using the theory of regions,” *IEEE Trans. on Robotics & Automation*, vol. 19, pp. 137–141, 2003.

[100] A. Nazeem, S. Reveliotis, Y. Wang, and S. Lafortune, “Designing maximally permissive deadlock avoidance policies for sequential resource allocation systems through classification theory: the linear case,” *IEEE Trans. on Automatic Control*, vol. 56, pp. 1818–1833, 2011.

[101] A. Nazeem and S. Reveliotis, “Designing maximally permissive deadlock avoidance policies for sequential resource allocation systems through classification theory: the non-linear case,” *IEEE Trans. on Automatic Control*, vol. 57, pp. 1670–1684, 2012.

[102] R. Cordone, A. Nazeem, L. Piroddi, and S. Reveliotis, “Designing optimal deadlock avoidance policies for sequential resource allocation systems through classification theory: existence results and customized algorithms,” *IEEE Trans. on Automatic Control*, vol. 58, no. 11, pp. 2772–2787, 2012.

[103] S. Reveliotis and A. Nazeem, “Optimal linear separation of the safe and unsafe subspaces of sequential RAS as a set-covering problem: algorithmic procedures and geometric insights,” *SIAM Journal on Control and Optimization*, vol. 51, pp. 1707–1726, 2013.

[104] A. Nazeem, “Designing compact and maximally permissive deadlock avoidance policies for complex resource allocation systems through classification theory,” Ph.D. dissertation, Georgia Tech, Atlanta, GA, 2012.

[105] A. Nazeem and S. Reveliotis, “Efficient enumeration of minimal unsafe states in complex resource allocation systems,” *IEEE Trans. on Automation Science and Engineering*, vol. 11, pp. 111–124, 2014.

[106] S. Reveliotis, “Algebraic deadlock avoidance policies for sequential resource allocation systems,” in *Facility Logistics: Approaches and Solutions to Next Generation Challenges*, M. Lahmar, Ed. Auerbach Publications, 2007, pp. 235–289.

[107] J. Campos, C. Seatzu, and X. Xie, Eds., *Formal Methods in Manufacturing.* CRC Press, 2014.

[108] B. Bollig and I. Wegener, “Improving the Variable Ordering of OBDDs Is NP-Complete,” *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 993–1002, 1996.

[109] C. E. Shannon, “A Mathematical Theory of Communication,” *The Bell System Technical Journal*, vol. 27, pp. 379–423,625–656, 1948.

[110] H. Andersen, “An introduction to binary decision diagrams,” Department of Information Technology, Technical University of Denmark, Tech. Rep., 1999.

[111] B. Gaudin, “Efficient solution for the State Avoidance Control Problem on Concurrent Systems using a disjunctive architecture,” in *Proceedings of the 8th international Workshop on Discrete Event Systems, WODES’06*, Ann Arbor, MI, USA, Jul. 2006, pp. 70–75.

[112] C. Lewerentz and T. Lindner, Eds., *Formal Development of Reactive Systems—Case Study Production Cell*, ser. Lecture Notes in Computer Science. Springer, 1995, vol. 891, ch. II, pp. 7–19.

[113] M. Fabian, Z. Fei, S. Miremadi, B. Lennartson, and K. Åkesson, “Supervisory control of manufacturing systems using extended finite automata,” in *Formal Methods in Manufacturing*, J. Campos, C. Seatzu, and X. Xie, Eds. CRC Press / Taylor and Francis, 2014, pp. 295–314.

[114] C. Ma, J. Ma, and W. Wonham, *Nonblocking Supervisory Control of State Tree Structures*, ser. Lecture Notes in Control and Information Sciences. Springer, 2005.

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

[116] T. Kelly, Y. Wang, S. Lafortune, and S. Mahlke, “Eliminating concurrency bugs with control engineering,” *Computer*, vol. 42, no. 12, pp. 52–60, Dec 2009.

[117] A. Nazeem and S. Reveliotis, “Maximally permissive deadlock avoidance for resource allocation systems with R/W-locks,” in *Proceedings of WODES 2012.* IFAC, 2012.