THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

On Symbolic Analysis of Discrete Event Systems Modeled as Automata with Variables

Zhennan Fei

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

On Symbolic Analysis of Discrete Event Systems Modeled as Automata with Variables

Zhennan Fei

Technical report number: R007/2012
ISSN 1403-266X

Department of 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 2012

to my family

Abstract

In benefit of the current revolution in computer technology, nowadays, society is dependent on dedicated computer-aided systems more than ever to assist us in every aspect of daily life. Thereby, designing reliable control logic of those systems to avoid malfunctioning behavior is of importance.

At some certain level of abstraction, the dynamics of many computer-aided systems can be characterized by a set of states and the state evolution depends entirely on the occurrence of discrete events. Such dynamic systems are referred to as discrete event systems (DES), which is the main subject of this thesis.

Supervisory Control Theory (SCT) is a formal methodology for generating control function for DESs based on a model of an uncontrolled plant and specifications that the closed-loop system must fulfill. SCT makes it easier to handle changes for the system to be controlled. This is, for example, important for manufacturing systems where both the products to be produced and the production equipment may change frequently. With such a model-based framework, it is possible to use algorithms to generate large parts of the control logic.

Although SCT shows great promise to assist control engineers to create correct control functions, industrial acceptance has been limited so far. One of the main obstacles with SCT is the state-space explosion problem which arises from the failure of explicit enumerating and storing large number of states due to lack of memory. To alleviate this problem, a well-known strategy is to utilize compact data structures such Binary Decision Diagrams (BDDs) to efficiently represent set of states. By encoding system models, the computation of control functions can be carried out implicitly (symbolically).

In this thesis, DESs are modeled as Extended Finite Automata (EFAs), which are ordinary automata augmented with variables. By taking advantage of the EFA structure, this thesis presents a set of BDD-based algorithms and formal analysis for exploring the state-space of large-scale DESs. Specifically, by using one of the partitioning techniques, the algorithms partition the state-space of a considered DES into a set of BDDs according to the included events and explore them in an efficient and structural manner. The work presented in this thesis has been implemented and integrated into the SCT tool Supremica and the algorithm efficiency is demonstrated on a set of academic and industrial examples.

Keywords: Discrete Event Systems, Supervisory Control Theory, Extended Finite Automata, Binary Decision Diagrams, Partitioning Techniques

Acknowledgments

Truth be told, life is full of surprises and sometimes it just takes the breath away. I still remember the afternoon when Bengt called, telling me that I was offered to pursue my Ph.D. degree in Automation group. I was so happy, surprisingly. To be honest, I never expected that I would obtain so much from these two and half years. However, I truly did and even my life has been changed. Now, half of the journey has passed by, I would like to take this opportunity to express my heartfelt thanks to a number of people, without whom the thesis would never be possibly done.

Foremost, how can I ever express my thanks to Knut, my main supervisor? Your patient guidance, insightful suggestions and brilliant ideas are the most generous gifts on the whole thesis work. Thank you so much! Secondly, my great gratitude goes to Bengt, my co-supervisor for your unlimited support and encouragement. Thank Spyros for giving me such a great opportunity to visit you and have good discussions at Georgia Tech. Besides I also would like to thank all of my colleagues at the division, especially the DK members in the Automation group. You guys are amazing. Special thanks to Sahar for sharing feelings and exchanging life experiences as foreigners, and Alexey for constantly helping me solving problems about programming. I am also grateful to Sajed. I really appreciate the time of discussing our ideas in front of the white board(s). It is a great pleasure to work with you. At the administrative level, I would like to thank our IT administrator Lars and secretaries Madeleine and Christine for always being so kind and helpful.

Moreover, I am indebted to my family and all the friends in Sweden and far away from China (6385.71 km) for believing in me and backing me up for good and bad times. At last, but definitely not at least, I want to express my greatest gratitude to Xuan. You mean the most to me.

Zhennan Fei
Atlanta, May 2012

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, *will appear as a book chapter in volume 0271 of the Communications in Computer and Information Science (CCIS) series, 2012*.

Paper 2 Zhennan Fei, Knut Åkesson, Bengt Lennartson: Symbolic Reachability Computation using the Disjunctive Partitioning Technique in Supervisory Control Theory, *Proceedings of the 2011 IEEE Conference on Robotics and Automation (ICRA 2011), pp. 4364-4369, 2012*.

Paper 3 Zhennan Fei, Sajed Miremadi, Knut Åkesson, Bengt Lennartson: Efficient Supervisory Synthesis to Large-Scale Discrete Event Systems Modeled as Extended Finite Automata, *Submitted to for possible journal publication, 2012*.

Paper 4 Zhennan Fei, Sajed Miremadi, Knut Åkesson: Modeling Sequential Resource Allocation Systems using Extended Finite Automata, *Proceedings of the Seventh Annual IEEE Conference on Automation Science and Engineering (CASE 2011), pp. 444-449, 2011*.

The individual contributions of each paper are outlined in Chapter 5.

Other publications

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

Paper 5 Zhennan Fei, Sajed Miremadi, Knut Åkesson, Bengt Lennartson: A Symbolic Approach to Large-Scale Discrete Event Systems Modeled as Finite Automata with Variables, *Submitted to the Eighth Annual IEEE Conference on Automation Science and Engineering (CASE 2012), 2012*.

Paper 6 Sajed Miremadi, Zhennan Fei, Knut Åkesson, Bengt Lennartson: Symbolic Nonblocking Computation of Timed Discrete Event Systems, *Submitted to the* 51^{st} *Annual IEEE Conference on Decision and Control (CDC 2012), invited paper, 2012*.

Paper 7 Sajed Miremadi, Zhennan Fei, Knut Åkesson, Bengt Lennartson: Symbolic Nonblocking Computation of Timed Discrete Event Systems-Extended Version, *Manuscript to be submitted for possible journal publication* .

Paper 8 Zhennan Fei, Sajed Miremadi, Knut Åkesson, Bengt Lennartson: Efficient Symbolic Supervisory Synthesis and Guard Generation - Evaluating Partitioning Techniques for the State-space Exploration, *Proceedings of the 2011 International Conference on Agents and Artificial Intelligence (ICAART 2011): 106-115, 2011*.

Contents

Abstract **i**

Acknowledgments **iii**

List of Publications **v**

Contents **vii**

I Introductory Chapters

1 Introduction **1**
1.1 Contributions 4
1.2 Outline 4

2 Supervisory Control Theory **7**
2.1 Modeling Formalisms 9
2.1.1 Extended Finite Automata (EFAs) 9
2.1.2 Deterministic Finite Automata (DFAs) 12
2.1.3 Composition of Subsystems 12
2.2 Supervisory Synthesis 14
2.2.1 Desired Properties of Supervisors 14
2.2.2 Safe-State Algorithm 15
2.2.3 Representing Supervisors with Guards 17

3 Symbolic Computation of Supervisors **19**
3.1 Binary Decision Diagrams 20
3.1.1 Representation of Models 21
3.1.2 BDD-Based Safe-State Algorithm 22
3.2 Symbolic Synthesis for EFAs 24
3.2.1 The Monolithic Approach 24
3.2.2 The Partitioning Approach 26
3.2.3 Structural State-Space Exploration 29

4 User Cases **33**
4.1 Modeling Sequential Resource Allocation Systems using EFAs . 33
4.2 Experimental Results 40

5 Summary of Included Papers **45**

6 Conclusions and Future Work **47**

References **49**

II Included Papers

Paper 1 Symbolic State-Space Exploration and Guard Generation in Supervisory Control Theory **59**
1 Introduction 59
2 Motivating Example 61
3 Preliminaries 63
3.1 Supervisory Control Theory 63
3.2 Binary Decision Diagrams (BDD) 65
4 BDD-Based Partitioning Computation 66
4.1 Efficient State Space Search 66
4.2 Workset Based Strategies 68
5 Supervisor as Guards 70
5.1 Computation of the Basic State Sets 71
5.2 Guard Generation 71
6 Case Studies 72
6.1 Benchmark Examples 72
6.2 Approach Evaluation 73
7 Conclusions 74
References 75

Paper 2 Symbolic Reachability Computation using the Disjunctive Partitioning Technique in Supervisory Control Theory **81**
1 Introduction 81
2 Preliminaries 82
2.1 Deterministic Finite Automata 82
2.2 Supervisory Control theory 83
2.3 Binary Decision Diagrams 85
3 Efficient Reachability Computation 85
3.1 Partitioning of the Full Synchronous Composition . . . 86
3.2 Workset Strategy 88

4 Algorithm Efficiency 93
5 Conclusions 94
References 94

Paper 3 Efficient Supervisory Synthesis to Large-Scale Discrete Event Systems Modeled as Extended Finite Automata 99
1 Introduction 99
2 Preliminaries 102
2.1 Extended Finite Automata 102
2.2 Binary Decision Diagrams 105
2.3 Supervisory Control Theory 107
3 A Motivation Example 108
4 Partitioning of the full synchronous composition 111
5 Efficient Reachability Computation 117
5.1 An Event-based Forward Reachability Algorithm 118
5.2 The Proposed Algorithm is Correct 120
5.3 Several Variants Of The Algorithm 121
6 Case Studies 122
6.1 Benchmark Examples 122
6.2 Results 126
7 Conclusions 128
References 128

Paper 4 Modeling Sequential Resource Allocation Systems using Extended Finite Automata 135
1 Introduction 135
2 Preliminaries 137
2.1 Conjunctive/Disjunctive Resource Allocation System . . 137
2.2 Extended Finite Automaton (EFA) 138
2.3 Supervisory Control Theory (SCT) 138
3 the proposed modeling approach 140
3.1 Model Single-Unit (SU) RAS 140
3.2 Model C/D RAS 142
3.3 Model the Abnormal Behavior 144
4 Case Studies 145
4.1 Benchmark Examples 146
5 Conclusions 147
6 Acknowledgement 148
References 148

Part I

Introductory Chapters

Chapter 1

Introduction

We are living in a fast changing era -- the era in which almost every aspect of daily life is being fundamentally changed with the current revolution in computer technology. From the very moment of waking up in the morning, human beings are surrounded themselves with diversely dedicated hardware and software systems, e.g., mobile phones, transportation facilities, communication tools, medical devices.

As such systems are widely used, our reliance on the functioning of them is growing unprecedentedly. As a consequence, failure, is unacceptable. We are annoyed when our mobile phones malfunction, or when two robots in a manufacturing system moves unexpectedly and collide with each other. These errors do not threaten our lives but have substantially financial consequences for the manufacturer. In the early nineties, a bug in Intel's Pentium II floating-point division unit [1] caused a loss of about 455 million US dollars to replace faulty processors. It also severely damaged Intel's reputation as a reliable chip manufacturer. It is not only about money, but also safety. Errors can be catastrophic too. A recent example of such a failure is the Ariane-5 rocket [2], which 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. This conversion was not protected. 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 of critical applications necessitates the development of formal methods for rigorously modeling systems and accurately assessing their functional properties. With respect to system modeling, classical control theory deals with systems whose behavior can be formulated as a set of differential or difference equations. Such systems are *time-driven*, since the system equations are parameterized by time. On the other hand, at a certain level

of abstraction, the behavior of systems such as automated manufacturing systems, computer networks and embedded systems can be profitably 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 time, is referred to as a *discrete event system (DES)* [3]. A comprehensible example that can be modeled a DES is the traffic light system. At a certain level of abstraction, the traffic light system consists of three states denoting either green, amber or red, as well as a series of events modeling the alternation between the lights, e.g., the light turns green to amber.

During the last two decades, research in formal methods has led to some promising verification and synthesis approaches that ease the burden of designing reliable systems. One such approach towards the correctness of computer-based systems is *model checking* [4, 5], a formal verification technique allowing for desired behavioral properties of a given system to be verified on the basis of a model of the system automatically. While verification generally terminates with a *yes* or *no* answer to the satisfiability of the system with respect to properties, synthesis goes further and directly generates such a desirable system where properties are fulfilled. In 1987, Ramadge and Wonham proposed a model-based framework called *Supervisory Control Theory (SCT)* [6, 7], which automatically generates a correct *control function*, referred to as the *supervisor* for discrete event systems. Given a model of the system to be controlled, the *plant*, and a model of the desired behavior of the controlled system, the *specification*, the supervisor, can be automatically generated, *synthesized*, to control the plant according to the specification. In SCT, DESs are often formally represented *deterministic finite automata* (DFAs) [8] or *extended finite automata* (EFAs) [9] that are DFAs augmented with variables. Besides, it is assumed that all of events are generated from the plant and the resultant supervisor is minimally restrictive, meaning that the plant is given the greatest amount of freedom to generate events without dissatisfying the specification.

In today's industry, designing and implementing control functions is still carried out manually through a series of verification steps, which is both time-consuming and error-prone. By applying SCT, the control function can be automatically generated in an algorithmic and systematical way, and thus the reliability of resultant control functions is enhanced. SCT has been applied to different research areas, such as manufacturing systems [10, 11], chemical batch processing systems [12, 13], and communication systems [14].

Similar to model checking, synthesis needs explore all possible system states. In the standard approach, these states are enumerated and stored explicitly during verification or synthesis. However, nowadays, the magnitude of systems, as well as their complexity, grows rapidly. They are not stand-alone, but are typically embedded in a larger context, connecting and interacting with several other

systems. To synthesize the supervisor for such complex applications, explicit enumeration quickly becomes a serious impediment and might suffer from the *state-space explosion* problem due to lack of available memory. In general, two well-known strategies can be utilized to alleviate the state-space problem and thus handle complex industrial applications: *compositional approach* or *symbolic approach*. Regarding the former, by exploiting the modularity structure of automata and applying a series of abstractions, the synthesis task can be substantially improved [15, 16, 17]. The latter strategy, the main focus in this thesis, is achieve by the introduction of Binary Decision Diagrams (BDDs) [18, 19], compact and operation-efficient data structures for manipulating Boolean functions. After encoding system states and associated events, i.e. system transitions, the explicit supervisory synthesis algorithm can be translated to the corresponding counterpart where the supervisory synthesis can be carried out implicitly or symbolically in BDDs. However, the symbolic computation is not a silver bullet. Transforming from the explicit state-space traversal algorithm into a BDD-based computation scheme does not guarantee that the algorithm will become remarkably efficient. During the synthesis computation, the state-space explosion problem still occurs due to the huge size of BDD nodes, even though the resultant BDD representation is manageable. Regarding this issue, inspired from the model checking community, *partitioning techniques* [20] come in handy for the reduction of intermediate size of BDDs. The idea is to split the monolithic transition under full synchronous composition into a set of less complex components with some logic connection in between. According to some heuristics, one small component is selected at a time for participating the symbolic computation and thus the chance of causing the explosion, comparing to having a huge monolithic transition of a given DES, is reduced. From prior work, such techniques have been applied to DESs represented as DFAs [21, 22] and relatively complicated applications can be solved that were not possible with explicit enumeration methods. In the light of the remarks above, the main objectives in this thesis is to enhance the results from prior work and meanwhile, apply partitioning techniques to DESs modeled as EFAs where the analysis is more difficult due to the introduction of variables.

With the computation efficiency brought by the application of BDDs, it is possible to synthesize supervisors of complex industrial examples with significantly huge state-space. Meanwhile, another problem arises from the interpretation of supervisors represented symbolically. Since the original models (DFAs or EFAs) have been reformulated and encoded, it is cumbersome for the users to relate each state with the corresponding BDD variables. Therefore, it is more convenient and natural to represent the supervisor in a form similar to the models. In [23], a promising approach is presented, where a set of minimal and tractable logic expressions, referred to as *guards*, are extracted from the supervisor and

attached to the original models. The guard generation approach [23] is used to generate guards from supervisors as soon as they are symbolically computed.

As a supplement, the work presented in this thesis has been integrated into the modeling and synthesis framework from prior work. Overall, the whole framework provides the convenience for users to model systems and obtain control functions in the same model domain. All symbolic computations are performed efficiently by BDDs, which are transparent and the only interface users deal with is the modeling formalism provide by the framework. The presented symbolic algorithms have been implemented in the SCT tool Supremica [24, 25], and the efficiency is demonstrated on a set of academic and industrial applications.

1.1 Contributions

The contributions of this thesis are:

- Identified a problem with a previously published algorithm for supervisory synthesis. Suggested a modification to the algorithm and proved the correctness.
- Adapted the symbolic SCT algorithm to take advantage of the EFA structure and thus avoiding a costly pre-processing step that translated EFA to DFA models.
- Suggested and benchmarked several ways to partition the symbolic representations of the DFA and EFA models such that the intermediate blow up of internal nodes in the BDD representation are kept small.
- Proposed an approach to modeling and efficient synthesis of control logic for resource allocations systems with routing flexibility and error handling.
- Adapted the suggested SCT algorithms to the guard generation procedure, making it more applicable for industrially interesting applications.
- Implemented all the suggested algorithms in Supremica.

1.2 Outline

The scope of this thesis is constituted by two parts. Serving as a general introduction to this field, Part I provides the preliminaries and puts the papers into the context. Part II contains the papers, authored or co-authored. Regarding Part I, Chapter 2 describes several modeling formalisms, the core concepts in SCT and the main synthesis algorithm. Chapter 3 focuses on the symbolic computation

where a set of symbolic algorithms are described and discussed. In Chapter 4, an approach to modeling sequential resource allocation systems is firstly detailed. Then we show the experimental results by applying the symbolic algorithms in this thesis to a set of benchmark examples. A summary of the appended papers is provided in Chapter 5. Finally, some concluding remarks and future work are given in Chapter 6.

Chapter 2

Supervisory Control Theory

Supervisory Control Theory, SCT [6, 7, 26, 3], developed by Ramadge and Wonham in the 80's, is a formal framework for control and analysis of discrete event systems (DESs). In the theory, the plant which is the model of a given uncontrolled system, is assumed to spontaneously generate all physically possible events. An important feature of the SCT is the partitioning of events as controllable or uncontrollable. If Σ is an event set, then $\Sigma_c \subseteq \Sigma$ denotes the subset of controllable events and its complement $\Sigma_u \subseteq \Sigma$ the subset of uncontrollable events. Note that only controllable events can be prevented from occurring by a supervisor. For synthesis, a specification is needed. A specification is a description of the intended behavior of the closed-loop system. The task of the supervisor is to restrict the uncontrolled behavior of the plant such that the specification is fulfilled.

Given a system to be controlled, plant, and the intended behavior of the close-loop system, specification, the algorithm provided by SCT can automatically synthesize a supervisor which restricts the conduct of the plant to ensure that the close-loop system never violates the given specification. Figure 2.1 shows the interaction between the plant and the supervisor. By following the generated events from the plant, the supervisor only restricts the controllable events that can be generated in the current state of the plant. It is worth mentioning that, typically the supervisor is not unique. A class of supervisors that do not allow anything to happen also fulfills the specification in the sense that the plant is not allowed to do anything outside the specification. However, such supervisor is not useful. In SCT, supervisors are assumed to be *minimally restrictive*, which means plants are give 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. It has been proved that such supervisor exists and is unique with respect to a given plant and specification. Therefore, unless otherwise noted, in this work we will only consider synthesis of supervisors with minimally restrictive closed-loop behavior.

SCT shows great promise to help control engineers to create correct control functions for discrete event systems. Comparing to the current approach adopted in industry, where the generation of control functions is still carried out manually, SCT provides a formal and algorithmic methodology for analyzing properties of DESs and automatically generating reliable control functions. Unfortunately, industrial acceptance has been limited so far. One fact is that SCT problems are inherently hard to solve while industrial systems usually have high complexity. It has been well-know that computing the minimal restrictive supervisor is NP-hard. This implies in the worst case we cannot expect a solution that is faster than the brute-force solution. Therefore, there is a need to develop more intelligent algorithms.

Although SCT originates from the control engineering community, it does have some similarities with some well-known and widely accepted research areas in computer science, such as *planning* in Artificial Intelligence and *model checking*. In planning [27], given a set of feasible steps or operations together with one or more objectives, a plan is generated containing a series of valid operations that are capable of taking the model to the desired state. There is a difference between planning and SCT is that in SCT, a supervisor is required to be minimally restrictive by including every possible feasible solutions while in planning, the algorithm terminates when one solution is found. Nevertheless, both SCT and AI planning suffer from the same complexity problems. There are a number of researches dedicated to the improvement of planning algorithms. Some of the best known approaches can be applied to the context of SCT, such as partial-order planning [28], symbolic computation [29, 30].

As mentioned in the introduction, the major difference between SCT and modeling checking is that unlike model checking, where the goal is to verify whether a model satisfies a given specification, SCT yields a correct model which behaves according to the specification. In addition, in model checking the specification is usually a temporary logic formula, while in SCT, the modeling formalism such as automata is used to model both plant and specification. Because of the strong relation between SCT and model checking, in particular, symbolic model checking [31, 20, 32, 33], a number of promising approaches have been accommodated and applied in SCT. The approaches proposed in the thesis and attached papers are inspired by them.

In this chapter, a number of preliminaries and extensions in SCT such system modeling, synthesis algorithm and supervisor representation are introduced, which are used throughout the thesis.

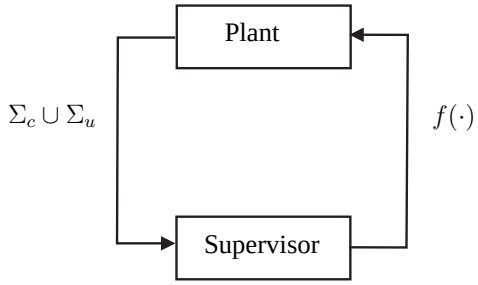


Figure 2.1: The interaction between plant and supervisor. Σ_c and Σ_u denote controllable and uncontrollable events respectively. $f(\cdot)$ is an event disabling function [3].

2.1 Modeling Formalisms

When it comes to representation and manipulation of discrete event systems, there are a number of modeling formalisms that can be used depending on the different objectives of analysis. In our work, to improve the expressiveness and compactness of system models, we use an extended variant of ordinary automata, where variables are introduced [9].

In general, a DES can be non-deterministic, which means that some transitions can occur because of some internal, non-observable behavior of the system. On the occurrence of a certain event in a certain state, the system may transit to different states at different occasions. For a deterministic system on the other hand, the next state is uniquely determined by the current state of the system and the event itself. In this thesis, the focus is on deterministic models.

2.1.1 Extended Finite Automata (EFAs)

An EFA, introduced in [9], is an augmentation of the ordinary finite automaton (FA) with guards predicates and actions functions. The guard predicates and actions are associated to the transitions of automata. A transition in an EFA is enabled if and only if its corresponding guard predicate is evaluated to `true`, and when a transition is taken, updating actions of a set of variables may follow. Guard predicates can be realized by their characteristic functions.

Definition 2.1.1 (Characteristic Function)**.** Let W be a finite set so that $W \subseteq U$, where U is the finite universal set. A *characteristic function* $\chi_W \colon U \to \mathbb{B}$ is defined by

$$\chi_W(a) = \begin{cases} 1 & \text{iff } a \in W \\ 0 & \text{iff } a \notin W \end{cases} .$$

Let n be the number of elements in U, in practice its elements are represented with numbers in $\mathbb{Z}_n$ or binary m-tuples in $\mathbb{B}^m (m = \lceil \log_2^n \rceil)$. For binary characteristic functions, an injective function $\theta \colon U \to \mathbb{B}^m$ is used to map the elements in U to elements in $\mathbb{B}^m$. In general, $\chi_W(a)$ is constructed as

$$\chi_W(a) = \bigvee_{w \in W} a \leftrightarrow \theta(w),$$

where $\leftrightarrow$ on two m-tuples v_1 and v_2 is defined as

$$v_1 \leftrightarrow v_2 \triangleq \bigwedge_{0 \leq i < m} (v_1^i \leftrightarrow v_2^i),$$

where v^i denotes the i:th element in the binary m-tuple v. As we will see later, characteristic functions can also be used to represent BDDs.

Definition 2.1.2 (Extended Finite Automata)**.** An extended finite automaton E is a 6-tuple

$$E = \langle L^E \times V, \Sigma^E, \mathcal{G}, \mathcal{A}, \rightarrow, (\ell_0^E, v_0) \rangle,$$

where:

- $L^E \times V$ is the extended finite set of states, denoted by Q, where L^E is a set of *locations* and V is the domain of definition of the *variables*;
- Σ^E is a non-empty finite set of events, called *alphabet*;
- $\mathcal{G} = \{\chi_W \mid W \in 2^V\}$ is the set of guard predicates over V;
- $\mathcal{A} = \{a \mid a \colon V \to V\}$ is a collection of action functions;
- $\rightarrow \subseteq L^E \times \Sigma^E \times \mathcal{G} \times \mathcal{A} \times L^E$ is the transition relation;
- $(\ell_0^E, v_0) \in L^E \times V$ is the initial state.

The finite set $V = V^1 \times \ldots \times V^n$ is the domain of definition of an n-tuple of variables $v = (v^1, \ldots, v^n)$ with the initial values $v_0 = (v_0^1, \ldots, v_0^n) \in V$. A *guard* $g(v)$ is a predicate over the variables that relate each element of V to either 1 (`true`) or 0 (`false`). Actions are written as

$$\acute{v} \colon = a(v) = (a^1(v), \ldots, a^n(v)), \text{where } \acute{v} \in V.$$

The symbol ξ is used to denote implicit actions that do not update the values of variables. For instance, if $a^i(v) = \xi$, it means that action a^i does not update variable v^i, i.e. $\acute{v}^i = v^i$.

The transition relation can be written as $\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}$, where $\ell, \acute{\ell} \in L, \sigma \in \Sigma, g \in \mathcal{G}$ and $a \in \mathcal{A}$. If g is absent, denoted by $\ell \xrightarrow{\sigma}_{a} \acute{\ell}$, it is assumed that

g always evaluates to `true`. If a is absent, denoted by $\ell \xrightarrow{\sigma}_{g} \acute{\ell}$, it is assumed that $a(v) = \Xi$, where Ξ is the vector notation for $(\xi, \xi, \ldots, \xi)$, indicating that no variable is updated during the transition.

For convenience, the states (locations and variable values) can be explicitly written in system transitions according to the following definition.

Definition 2.1.3 (Explicit State Transition Relation)**.** Let $E = \langle L^E \times V, \Sigma^E, \mapsto, (\ell_0^E, v_0)\rangle$ be an EFA. The explicit state transition relation of E is defined as

$$\begin{aligned} \mapsto_E \triangleq\ & \{(\ell^E, v, \sigma, \acute{\ell}^E, \acute{v}) \in L^E \times V \times \Sigma \times L^E \times V \mid \\ & \exists \ell^E \xrightarrow{\sigma}_{g/a} \acute{\ell}^E \colon v \in \mathsf{SAT}\mathcal{G}(g) \wedge (v, \acute{v} \in \mathsf{SAT}\mathcal{A}(a))\}, \end{aligned}$$

where v and $\acute{v}$ are the values of the variables before and after executing the transition, respectively; $\mathsf{SAT}\mathcal{G}$ denotes the set of variable assignments that satisfies the guard $g(v)$,

$$\mathsf{SAT}\mathcal{G}(g) \triangleq \{v \in V \mid v \vDash g\};$$

and $\mathsf{SAT}\mathcal{A}$ denotes the following set:

$$\mathsf{SAT}\mathcal{A}(a) \triangleq \{(v, \acute{v}) \in V \times V \mid \acute{v} = a(v)\}.$$

For brevity, we denote the explicit representation of a transition $\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}$ by $\mapsto_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}$. Besides, since we are interested in deterministic systems, we merely focus on deterministic EFAs which are defined as follows. In the sequel, for the sake of brevity, we simply write EFAs for deterministic EFAs.

Definition 2.1.4 (Deterministic EFA)**.** An EFA $E = \langle L^E \times V, \Sigma, \mapsto, (\ell_0^E, v_0)\rangle$ is deterministic if $(\ell^E, v) \overset{\sigma}{\mapsto} (\ell_1^E, v_1)$ and $(\ell^E, v) \overset{\sigma}{\mapsto} (\ell_2^E, v_2)$ always implies $(\ell_1^E, v_1) = (\ell_2^E, v_2)$.

Example 1. In this example, we model a mathematical strategy game, called stick-picking game by using EFAs. Supposing that there are five sticks on a table, two players take turn by remove one or two sticks from the table. The winner is the player that takes the last stick. Fig. 2.2 shows the EFA model for this game. The EFA contains two locations denoting the two players in the game. Each location has two outgoing transitions label by the events in correspondence with the two options: remove one or two sticks. Furthermore, one variable $sticks$ is declared to maintain the number of sticks on the table and the value is updated when players remove sticks by taking different transitions.

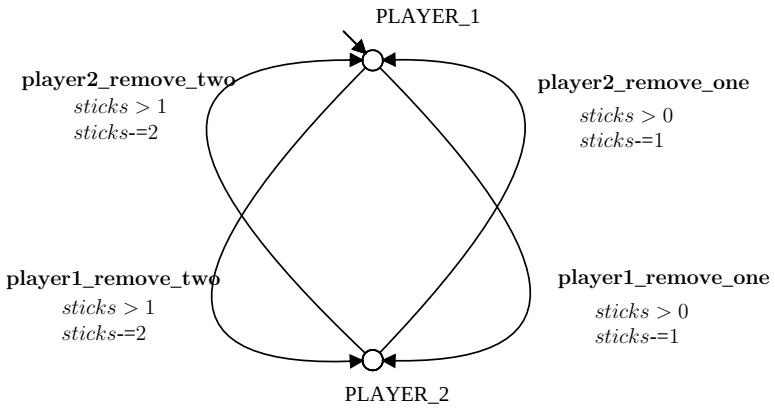


Figure 2.2: The EFA model for the stick-picking game

2.1.2 Deterministic Finite Automata (DFAs)

Deterministic Finite automata (DFAs) [8], from which EFAs are generalized, can be considered as a special case of EFAs where no variables are defined. Specifically, a deterministic finite automaton (DFA) can be reduced from the definition of deterministic EFA to a 4-tuple $\langle Q, \Sigma, \rightarrow, q_0 \rangle$, where Q and q_0 denote a finite set of states and the initial state of a DFA, corresponding to the finite location set and the initial location of an EFA where no variables are defined. Σ, same as EFA, is a non-empty finite set of events (the alphabet). For an EFA, since no variables are defined, guards and actions in the transition relation are thus omitted. This leads to $\rightarrow \subseteq Q \times \Sigma \times Q$, the state transition relation of DFA.

2.1.3 Composition of Subsystems

Usually, a considered DES can be modeled by a set of partially independent components (DFAs or EFAs). The total system is a *composition* of these subsystems. The composition of two concurrent subsystems can be done in several different ways. In our work, *full synchronous composition* (FSC) (Definition 2.1.5) and its extension, *extended full synchronous composition* (EFSC) (Definition 2.1.6) are used.

Definition 2.1.5 (Full Synchronous Composition [34])**.** Let $A_k = \langle Q^{A_k}, \Sigma^{A_k}, \rightarrow_{A_k}, q_0^{A_k} \rangle, k = 1, 2$ be two DFAs. The Full Synchronous Composition (FSC) of A_1 and A_2 is

$$A_1 \parallel A_2 = \langle Q^{A_1} \times Q^{A_2}, \Sigma^{A_1} \cup \Sigma^{A_2}, \rightarrow, (q_0^{A_1}, q_0^{A_2}) \rangle$$

where the state transition relation $\rightarrow$ is defined as

$$\begin{cases} \langle (q^{A_1}, q^{A_2}), \sigma, (\acute{q}^{A_1}, \acute{q}^{A_2}) \rangle & \text{if } \langle q^{A_1}, \sigma, \acute{q}^{A_1} \rangle \in \rightarrow_{A_1} \wedge \langle q^{A_2}, \sigma, \acute{q}^{A_2} \rangle \in \rightarrow_{A_2} \\ \langle (q^{A_1}, q^{A_2}), \sigma, (\acute{q}^{A_1}, q^{A_2}) \rangle & \text{if } \langle q^{A_1}, \sigma, \acute{q}^{A_1} \rangle \in \rightarrow_{A_1} \wedge \sigma \notin \Sigma^{A_2} \\ \langle (q^{A_1}, q^{A_2}), \sigma, (q^{A_1}, \acute{q}^{A_2}) \rangle & \text{if } \langle q^{A_2}, \sigma, \acute{q}^{A_2} \rangle \in \rightarrow_{A_2} \wedge \sigma \notin \Sigma^{A_1} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Definition 2.1.6 (Extended Full Synchronous Composition [9])**.** Let $E_k = \langle L^{E_k} \times V, \Sigma^{E_k}, \rightarrow_{E_k}, (\ell_0^{E_k}, v_0) \rangle, k = 1, 2$, be two EFAs with the shared variables $v = (v^1, \ldots, v^n)$. The Extended Full Synchronous Composition (EFSC) of E_1 and E_2 is

$$E_1 \parallel E_2 = \langle L^{E_1} \times L^{E_2} \times V, \Sigma^{E_1} \cup \Sigma^{E_2}, \rightarrow, (\ell_0^{E_1}, \ell_0^{E_2}, v_0) \rangle$$

where the state transition relation $\rightarrow$ is defined as

1. $(\ell^{E_1}, \ell^{E_2}) \xrightarrow{\sigma}_{g/a} (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \sigma \in \Sigma_1 \cap \Sigma_2$ if
   $\exists \ell^{E_1} \xrightarrow{\sigma}_{g_1/a_1} \acute{\ell}^{E_1} \in \rightarrow_{E_1}$ and
   $\exists \ell^{E_2} \xrightarrow{\sigma}_{g_2/a_2} \acute{\ell}^{E_2} \in \rightarrow_{E_2}$ such that:

   - $g = g_1 \wedge g_2$,
   - For $i = 1, \ldots, n$ and $\forall v \in V$:

$$a^i(v) = \begin{cases} a_1^i(v) & \text{if } a_1^i(v) = a_2^i(v) \\ a_1^i(v) & \text{if } a_2^i(v) = \xi \\ a_2^i(v) & \text{if } a_1^i(v) = \xi \\ v^i & \text{otherwise} \end{cases}$$

2. $(\ell^{E_1}, \ell^{E_2}) \xrightarrow{\sigma}_{g/a} (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \sigma \in \Sigma_1 \backslash \Sigma_2$ if
   $(\ell^{E_1}, \sigma, g, a, \ell_{E_1}) \in \rightarrow_{E_1}$ and $\ell^{E_2} = \acute{\ell}^{E_2}$;

3. $(\ell^{E_1}, \ell^{E_2}) \xrightarrow{\sigma}_{g/a} (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \sigma \in \Sigma_2 \backslash \Sigma_1$ if
   $(\ell^{E_2}, \sigma, g, a, \ell_{E_2}) \in \rightarrow_{E_2}$ and $\ell^{E_1} = \acute{\ell}^{E_1}$.

Note that, in the case where the action functions of E_1 and E_2 explicitly try to update a shared variable to different values, we assume that the variable is not updated. Such situation is usually a consequence of bad modeling, and thus it is reasonable to inform the user by a message rather than disabling the transition. However, we wanted the EFSC to always be well defined and thus we made the decision to keep the value in this situation.

Normally, a plant is described by a number of sub-plants $P_1, \ldots, P_l$. To obtain the plant P, FSC or EFSC can be used and thus the total plant P is defined by $P_1 \parallel \ldots \parallel P_l$. Similarly, if the specification is described as a series of sub-specifications, $Sp = Sp_1 \parallel \ldots \parallel Sp_m$. Additionally, some states of an automaton which is typically a (sub-)specification, are considered as *marked*

states, denoted by Q_m. The marked states are these states that can be reached from the initial state. The set of marked states of a composed automaton $E_1 \parallel E_2$ is the Cartesian product of the corresponding sets of marked states. Note that for EFAs, a state is marked only if the location as well as the values of variables are both marked. Besides, some states can be specified as *forbidden states*, Q_{ex}, which are the states that should be forbidden. The set of forbidden states of a composed automaton $E_1 \parallel E_2$ is $Q_{ex}^{E_1} \times Q^{E_2} \cup Q^{E_1} \times Q_{ex}^{E_2}$.

2.2 Supervisory Synthesis

As mentioned earlier, the supervisor to be computed is assumed to minimally restrictive meaning that the plant is give the greatest amount of freedom to generate events and the resultant supervisor only disables certain controllable events when necessary. To synthesize such a supervisor, the closed-loop system $P \parallel Sp$ is firstly computed, which we refer to as S_0 in the sequel. During the synthesis procedure, some states are identified either blocking or uncontrollable, referred to as forbidden states, which should be excluded from S_0 in order to obtain the safe states, i.e. the states belonging to the supervisor.

2.2.1 Desired Properties of Supervisors

In addition to the basic property that the plant P under the control of the superior S should fulfill the given specification Sp, typically, there are two additional properties [6, 7] which supervisors are desired to have:

1. Non-blocking.
2. Controllable with respect to the plant P.

A non-blocking automaton is an automaton where at least one of the marked states Q_m should be reached by every reachable state. This is a *liveness* property where *unmarked deadlock* and *livelock* states should be avoided. A state with no outgoing transitions is called a *deadlock* state. The situation where a system enters a set of states that are strongly connected, but without any transition going out of the set, is called *livelock*. The union of unmarked deadlock and livelock states of an automaton A are referred to *blocking* states.

Definition 2.2.1 (Blocking States Q_{bl})**.** Given an automaton A, which is either a DFA or an EFA, the blocking states Q_{bl} of A can be defined as follows.

- A is a DFA, then

$$Q_{bl} = \{q \in Q_{reach}^{A} \mid \nexists \acute{q} \in Q_m \Rightarrow \langle q, \sigma, \acute{q} \rangle \in \rightarrow_A\}$$

where Q^A_{reach} denotes all the states in A which can reached by the initial state q^A_0 and $\rightarrow_A$ is the state transition relation of A.

- A is an EFA, then

$$Q_{bl} = \{(\ell, v) \in Q^A_{reach} \mid \nexists(\acute{\ell}, \acute{v}) \in L_m \times V_m \Rightarrow \langle \ell, v, \sigma, \acute{\ell}, \acute{v}\rangle \in \mapsto_A\}$$

where Q^A_{reach} denotes all the states in A which can be reached by the initial state (ℓ^A_0, v_0) and $\mapsto_A$ is the explicit transition relation of the EFA A. L_m and V_m denote the marked locations and values of variables.

The controllability is a safety property, saying that a supervisor is controllable with respect to the plant if the supervisor is able to follow all uncontrollable events that generated by the plant. Precisely, a supervisor S is controllable with respect to the plant P if, from any reachable state in $S \parallel P$, an enabled uncontrollable event σ_u in P is either enabled in S as well or σ_u is not included in the alphabet of S.

Definition 2.2.2 (Uncontrollable States Q_{uc})**.** Given a supervisor S and a plant P, if S is not controllable with respect to P, the corresponding uncontrollable states, denoted by Q_{uc}, can be defined as follows.

- Both P and S are DFAs, then

$$\begin{aligned} Q_{uc} = \{\langle q^P, q^S\rangle \in Q^{P\|S}_{reach} \mid \exists \sigma_u \in \Sigma^P_u \cap \Sigma^S \\ \Rightarrow \langle q^P, \sigma_u, \acute{q}^P\rangle \in \rightarrow_P \wedge \langle q^S, \sigma_u \acute{q}^S\rangle \notin \rightarrow_S\} \end{aligned}$$

where Σ^P_u is the uncontrollable event set of P. $\rightarrow_P$ and $\rightarrow_S$ are the state transition relations of P and S.

- Either P or S is an EFA, then

$$\begin{aligned} Q_{uc} = \{(\ell^P, \ell^S, v) \in Q^{P\|S}_{reach} \mid \exists \sigma_u \in \Sigma^P_u \cap \Sigma^S \\ \Rightarrow (\ell^P, v, \sigma_u, \acute{\ell}^P, \acute{v}) \in \mapsto_P \wedge (\ell^S, v, \sigma_u, \acute{\ell}^S, \acute{v}) \notin \mapsto_S\} \end{aligned}$$

where Σ^P_u is the uncontrollable event set of P. $\mapsto_P$ and $\mapsto_S$ are the explicit transition relations of P and S.

2.2.2 Safe-State Algorithm

The algorithm synthesizes the supervisor by first building the closed-loop $S_0 = P \parallel Sp$, then removing forbidden states from Q^{S_0} until the remaining safe states are both nonblocking and controllable. In this thesis, the synthesis algorithm that is used to calculate the maximally restrictive supervisor is called the *safe-state-synthesis*, introduced in [22].

As Algorithm 1 shows, given a set of forbidden states Q_x that could either be explicit forbidden states or forbidden states due to the controllability, the algorithm computes the set of safe states Q^S by iteratively removing the blocking states (Algorithm 2) and the uncontrollable states (Algorithm 3). Note that after the termination of the algorithm, not all of the safe states are reachable from the initial state. Therefore, a forward reachability search (Algorithm 4) is needed to exclude the safe states which are not reachable.

Algorithm 1: Safe-State Synthesis

Input: Q_x and Q^{S_0}
Output: Q^S
begin
1 $i := 0$;
2 $Q_x^0 := Q_x$;
repeat
3 $i := i + 1$;
4 $Q' := \texttt{RestrictedBackward}\,(Q_m, Q_x^{i-1})$;
5 $Q'' := \texttt{UncontrollableBackward}\,(Q^{S_0} - Q')$;
6 $Q_x^i := Q_x^{i-1} \cup Q''$;
until $Q_x^i = Q_x^{i-1}$;
7 $Q^S := Q^{S_0} - Q_x^i$;
end

Algorithm 2: Restricted Backward

Input: Q_m and Q_x
Output: Q_{co}
begin
1 $j := 0$;
2 $Q^0 := Q_m - Q_x$;
repeat
3 $j := j + 1$;
4 $Q^j := Q^{j-1} \cup \{q \mid \exists \acute{q} \in Q^{S_0}, \exists \sigma \in \Sigma^{S_0} \Rightarrow \langle q, \sigma, \acute{q} \rangle \in \rightarrow_{S_0}\} - Q_x$;
until $Q^j = Q^{j-1}$;
5 $Q_{co} := Q^j$;
end

Algorithm 3: Uncontrollable Backward

Input: Q_x
Output: Q_{ex}
begin
1 $k := 0$;
2 $Q^0 := Q_x$;
repeat
3 $k := k + 1$;
4 $Q^k := Q^{k-1} \cup \{q \mid \exists \acute{q} \in Q^{S_0}, \exists \sigma_u \in \Sigma_u^{S_0} \Rightarrow \langle q, \sigma_u, \acute{q} \rangle \in \rightarrow_{S_0}\}$;
until $Q^k = Q^{k-1}$;
5 $Q_{ex} := Q^k$;
end

Algorithm 4: Restricted Forward

Input: q_0 and Q_x
Output: Q_r
begin
1 $l := 0$;
2 $Q^0 := \{q_0\}$;
repeat
3 $l := l + 1$;
4 $Q^l := Q^{l-1} \cup \{\acute{q} \mid \exists q \in Q^{S_0}, \exists \sigma \in \Sigma^{S_0} \Rightarrow \langle q, \sigma, \acute{q} \rangle \in \rightarrow_{S_0}\} - Q_x$;
until $Q^l = Q^{l-1}$;
5 $Q_r := Q^l$;
end

2.2.3 Representing Supervisors with Guards

When designing the supervisor of a system, a typical issue is how to realize such a control function efficiently and represent it appropriately. The standard approach [6], or the safe-state approach above, synthesize the supervisor by first building the closed-loop system and then explicitly enumerate all states that can be reached. The problem of computing and representing supervisors in such monolithic and explicit way is that for large and complex systems, supervisors, with potential huge numbers of states, become intractable and might cause the *state-space explosion* problem.

An alternative approach to handle the computation of supervisors, is to use *binary decision diagrams* (BDDs) [18, 35, 36] to encode the transition relation of the closed-loop system and express the supervisor symbolically [37, 22, 38, 39, 40]. The discussion of BDDs and the applications in SCT will be detailed in Chapter 3. Here we suppose that the supervisor of a system has been computed

and represented by BDDs. Meanwhile, another problem is arising from the BDD representation of the resultant supervisor. Since the original models have been reformulated and encoded, it is cumbersome for the users to relate each state with the corresponding BDD variables. Therefore, it is more convenient and natural to represent the supervisor in a form similar to the models.

In [23], an approach, called the *guard generation procedure*, was presented. Being dependent on three kinds of state sets, the guard generation procedure extracts a set of conditional propositional formulae, referred to as *guards* indicating under which conditions the event can be executed without violating the specifications. These guards are then attached to the corresponding transitions of the original models, which results in a modular representation of supervisors.

Concerning the states that are retained or removed after the synthesis process, the states that enable an arbitrary event σ can be divided into three basic state sets: forbidden state set, allowed state set and don't care state set.

- The forbidden state set, denoted by $Q^{\sigma}_{\mathbf{f}}$, is the set of states in the supervisor where the execution of σ is defined for S_0, but not for the supervisor.
- The allowed state set, denoted by $Q^{\sigma}_{\mathbf{a}}$, is the set of states in the supervisor where the execution of σ is defined for the supervisor. In other word, for each event σ in S_0's alphabet, $Q^{\sigma}_{\mathbf{a}}$ represents the set of states where event σ must be allowed to be executed in order to end up in states belonging to the supervisor.
- The don't care state set, $Q^{\sigma}_{\mathbf{dc}}$ can be defined as the complement of the union of $Q^{\sigma}_{\mathbf{a}}$ and $Q^{\sigma}_{\mathbf{f}}$.

$$Q^{\sigma}_{\mathbf{dc}} = C(Q^{\sigma}_{\mathbf{a}} \cup Q^{\sigma}_{\mathbf{f}})$$

Based on the basic state sets, guards can be extracted, expressing under which conditions the events can be executed without violating the specifications. In addition, by applying minimization methods of Boolean functions (utilizing the don't care state set) and certain heuristics, the generated guards can be simplified. The detailed exposition is however beyond the scope of the thesis and can be found in [23].

Chapter 3

Symbolic Computation of Supervisors

The *state-space explosion* problem is one of the typical problems which prevent SCT from having a major industrial breakthrough. As mentioned earlier in Chapter 2, the prerequisite for synthesizing the supervisor of a system is the construction of the closed-loop system by performing the full synchronization composition on a set of modular models interacting with each other by common events. Due to the limitation of available memory, synchronizing multiple automata might fail if the consider system has a potential huge state space, e.g., millions of states.

Since the state-space explosion problem results from the failure of explicit enumeration of a large number of reachable states during synchronization, an alternative approach, inspired from model checking, is to represent states and transitions *symbolically*, or implicitly by using *binary decision diagrams* (BDDs) [18, 35, 36]. The main difference between explicit and implicit representation is that in the former approach, states are manipulated individually while in the latter one, set of states are manipulated at the same time. In addition, symbolic operations are carried out more efficiently compared to the explicit operations.

In this chapter, first a general symbolic computation approach using BDDs is described, including the representation of the transition relation of an EFA and the translation of safe-state algorithm to the corresponding symbolic counterpart. Following the symbolic discussion, two approaches to constructing the transition relation of synchronized EFAs, namely the monolithic and partitioning approaches, are described.

3.1 Binary Decision Diagrams

Binary Decision Diagrams (BDDs) [18, 19] are powerful data structures for representing Boolean functions. For large systems where the number of states grows exponentially, BDDs can improve the efficiency of set and Boolean operations performed on the state sets dramatically [22, 31, 41, 42].

Definition 3.1.1 (Binary Decision Diagrams)**.** Given a set of Boolean variables B, a BDD is a Boolean function $h\colon 2^B \to \{0,1\}$, which can be expressed using Shannon's decomposition [43]:

$$h = (\neg b_j \wedge h|_{b_j=0} \vee (b_j \wedge h|_{b_j=1}))\ \ b_j \in B$$

where $h\ |_{b_j=0}$ and $h\ |_{b_j=1}$ refer to assignment 0 and 1 to all occurrences of the Boolean variable b_j, respectively. A BDD is represented as a directed acyclic graph (DAG), which consists of two types of nodes: *decision nodes* and *terminal nodes*. A terminal node can either be *0-terminal* or *1-terminal*. Each decision node is labeled by a Boolean variable and has two edges to its *low-child* and *high-child*. The low- and high-child corresponds to the cases in the above equation where b_j is 0 (graphically represented by a *dotted* line) and 1 (graphically represented by a *solid* line), respectively. The *size* of a BDD refers to the number of decision nodes.

A BDD is *Ordered* (OBDD) if on all paths through the graph the variables respect a given linear order $b_1 \prec b_2 \prec \ldots \prec b_n$. In addition, an OBDD is *Reduced* (ROBDD) if [36]

1. (**uniqueness**) no two distinct nodes and have the same variable name and low- and high-child;

2. (**non-redundant tests**) no variable node has identical low- and high-child.

In [19], Randal Bryant prove the *canonicity* of ROBDDs. That is to say, let B and B' be two ROBDDs representing a given function f. Then B and B' are identical. This is the single most important property of ROBDDs, which allows us to perform tests such as equality, satisfiability efficiently. Today, ROBDDs are used in a number of model checking tools for verifying system models. Note that on the following, all BDDs are assumed to be ordered and reduced BDDs (ROBDDs).

The power of BDDs lies in their simplicity and efficiency to perform binary operations. A binary operator $\mathtt{op}$ between two BDDs h and g can be computed as

$$h\ \mathtt{op}\ g = [\neg b_j \wedge (h|_{b_j=0}\ \mathtt{op}\ g|_{b_j=0})] \vee [b_j \wedge (h|_{b_j=1}\ \mathtt{op}\ g|_{b_j=1})].$$

Most of the operations used when manipulating BDDs are linear in the product of the sizes of the operand BDDs. The main exception is the relational product operation, i.e., `relprod`. The relational product can be implemented with one conjunction and a series of existential quantifications, however the time complexity in the worst case is exponential. Moreover, it is well-known that the variable ordering impacts the size of the BDD dramatically, however, finding an optimal variable ordering of a BDD is an NP-complete problem [44]. In this thesis, a static BDD variable ordering is computed based on the method presented in [45]. In this method, the variable ordering is influenced by the ordering of interacting automata based on weighted search in the Process Communication Graph (PCG) [45]. A PCG for a set of automata is a weighted undirected graph, where the weight between two automata A_1 and A_2 is defined as $|\Sigma^{A_1} \cap \Sigma^{A_2}|$. With respect to the DESs modeled as finite automata with variables, such as EFAs, the interaction of automata is affected not only by the alphabets but also the global variables. To handle this issue, we add additional weights to the PCG by inspecting the guard predicates and actions functions of each transition. This is however beyond the scope.

For a more elaborate and verbose exposition of BDDs and the implementation of different operators, refer to [36, 35].

3.1.1 Representation of Models

By taking the advantage of characteristics functions 2.1.1, system models such as automaton can be symbolically represented by BDDs. Without loss of the generality, we focus on the symbolic representation of an EFA.

Given a EFA E, the characteristic function of the explicit state transition relation $\mapsto_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}$ can be constructed as:

$$\begin{aligned}
&\chi_{\mapsto_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}}(b^{V^1}, \ldots, b^{V^n}, \acute{b}^{V^1}, \ldots, \acute{b}^{V^n}, b^L, \acute{b}^L, b^\Sigma) = \\
&\quad \Bigg(\bigvee_{(v,\acute{v}) \in \mathsf{SAT}\mathcal{A}(a) | v \in \mathsf{SAT}\mathcal{G}(g)} \\
&\quad \bigwedge_{i=1}^{n} \big(b^{V^i} \leftrightarrow \theta(v^i) \ \wedge \ \acute{b}^{V^i} \leftrightarrow \theta(\acute{v}^i) \big) \Bigg) \ \wedge \\
&\quad b^L \leftrightarrow \theta(\ell) \ \wedge \ \acute{b}^L \leftrightarrow \theta(\acute{\ell}) \ \wedge \ b^\Sigma \leftrightarrow \theta(\sigma),
\end{aligned}$$

where b^Σ denotes the Boolean variables representing the alphabet while b^L and $\acute{b}^L$ are two different sets of Boolean variables representing the current and updated locations. For an EFA where n variables are defined, b^{V^i} and $\acute{b}^{V^i}$ denote the current and updated integer values of variables. In our framework, integers are

Table 3.1: Event and current location encoding for the EFA in Fig. 3.1

Event	$b_1^\Sigma b_0^\Sigma$	Location	b_0^L
`player1remove1`	0 0	$player1$	0
`player1remove2`	0 1	$player2$	1
`player2remove1`	1 0		
`player2remove2`	1 1		

represented in the two's complement system as array of BDDs [46]. In practice, it is very often that values of variables are defined as or updated to non-negative integers. In that case, for computational purposes, the BDD variable representing the sign-bit of a non-negative integer is omitted and thus only the magnitude is encoded. Besides, we assume that overflows are not allowed and thus we omit the cases where an overflow occurs. This is performed by removing all the variable assignments that result in values outside the domain of the variables. Consequently, the characteristic function of the transition relation of an EFA E will be

$$\chi_{\mapsto_E} = \bigvee_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell} \in \to_E} \chi_{\mapsto_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}} \wedge \bigwedge_{i=1}^{n} \chi_{V^i}(b^{V^i}) \wedge \bigwedge_{i=1}^{n} \chi_{V^i}(\acute{b}^{V^i}).$$

Example 2. Regarding the EFA model shown in Example 1 of Chapter 2, the corresponding symbolic transition relation is shown in Fig. 3.1. Note that the BDD does not contain the cases where $sticks < 0$ and $sticks > 5$. The BDD variables in the figure are labeled by numbers as follows

$$b^\Sigma = (b_1^\Sigma, b_0^\Sigma) = (1, 0), b^L = (b_0^L) = (2), \acute{b}^L = (\acute{b}_0^L) = (3),$$
$$b^{sticks} = (b_2^{sticks}, b_1^{sticks}, b_0^{sticks}) = (6, 5, 4),$$
$$\acute{b}^{sticks} = (\acute{b}_2^{sticks}, \acute{b}_1^{sticks}, \acute{b}_0^{sticks}) = (9, 8, 7).$$

where b_0 is the least significant bit. Table shows the encoding of events and locations. At it can be observed, the BDD in this example is larger than the EFA model. However, for large models the BDDs typically become much more compact.

3.1.2 BDD-Based Safe-State Algorithm

In Chapter 2, the synthesis algorithm, i.e., the safe-state algorithm is discussed on an explicit level. By making use of the characteristic functions, the corresponding symbolic safe-state and restricted backward reachability algorithms can be

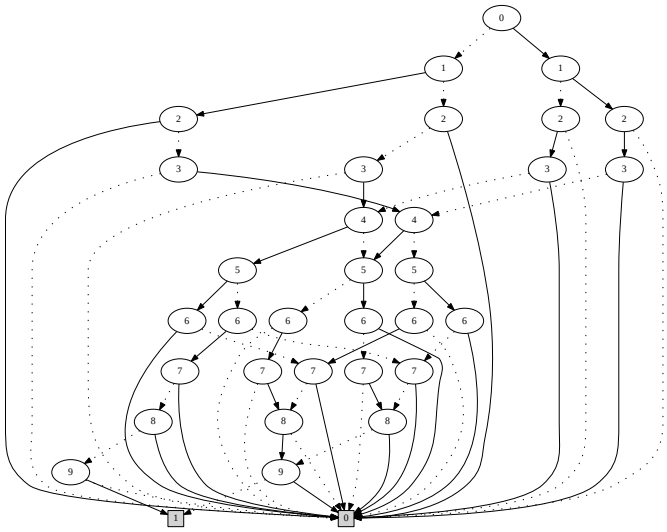


Figure 3.1: The Corresponding BDD for the Transition Relation of the EFA in Example 1.

obtained and are shown in Algorithm 5 and Algorithm 6 respectively. Note that $\rightarrow_{S_0}$ denotes the transition relation for DFAs or the explicit transition relation ($\mapsto_{S_0}$) for EFAs. The symbolic version of the restricted forward and uncontrollable reachability algorithms can be translated similarly. It can be observed states and transition relation are represented by their corresponding characteristic functions. Set operations, used to expand the state sets, are translated to logic operations.

Algorithm 5: B-Safe-State Synthesis

Input: χ_{Q_x} and $\chi_{Q^{S_0}}$
Output: χ_{Q^S}
begin
1 $i := 0$;
2 $\chi^0_{Q_x} := \chi_{Q_x}$;
repeat
3 $i := i + 1$;
4 $\chi_{Q'} :=$ `B-RestrictedBackward`$(\chi_{Q_m}, \chi^{i-1}_{Q_x})$;
5 $\chi_{Q''} :=$ `B-UncontrollableBackward`$(\chi_{Q^{S_0}} \wedge \neg\chi_{Q'})$;
6 $\chi^i_{Q_x} := \chi^{i-1}_{Q_x} \vee \neg\chi_{Q''}$;
until $\chi^i_{Q_x} \leftrightarrow \chi^{i-1}_{Q_x}$;
7 $\chi_{Q^S} := \chi_{Q^{S_0}} \wedge \neg\chi^i_{Q_x}$;
end

Algorithm 6: B-Restricted Backward

Input: χ_{Q_m} and χ_{Q_x}
Output: $\chi_{Q_{co}}$
begin
1 $j := 0$;
2 $\chi_Q^0 := \chi_{Q_m} \wedge \neg\chi_{Q_x}$;
repeat
3 $j := j + 1$;
4 $\chi_{Q'} := \exists\{b^\Sigma, \acute{b}^L, \acute{b}^V\}.(\chi_Q^{j-1} \wedge \chi_{\rightarrow_{S_0}})$;
5 $\chi_Q^j := \chi_Q^{j-1} \vee \chi_{Q'} \wedge \neg\chi_{Q_x}$;
until $\chi_Q^j \leftrightarrow \chi_Q^{j-1}$;
6 $\chi_{Q_{co}} := \chi_Q^j$;
end

At line 4 of Algorithm 6, `relprod` operation is used to calculate more co-reachable states from the marked states. Precisely, by performing the conjunction operation ($\wedge$) on the existing co-reachable state set χ_Q^{j-1} and $\rightarrow_{S_0}$, only the tuples where the target states belong to Q^{j-1} is retained. Then the new co-reachable state set can be obtained after existential quantifying the BDD variables for these target states ($\acute{b}^L$ and $\acute{b}^V$) and events (b^Σ). It should be mentioned that after the execution of line 4, $\chi_{Q'}$, the set of new states co-reached from Q_{j-1}, contains the source BDD variables, i.e., b^L and b^V. As an additional step, which is not shown in Algorithm 6, we need to replace those source variables with the corresponding target variables before merging them into the existing co-reachable state set.

3.2 Symbolic Synthesis for EFAs

At this moment, readers might ask the question: since the (extended) full synchronous composition is an explicit operation, how to acquire the symbolic representation of the (explicit) transition relation? In the upcoming sections, the question will be answered.

3.2.1 The Monolithic Approach

The characteristic function of the synchronized transition relation can be computed monolithically. For the sake of understandability and reachability, first the case that systems are modeled as DFAs is handled then we extend the approach to the extended case.

Definition 3.2.1 (Extended Transition Relation)**.** For $N \geq 2$ DFAs, $A_1, \ldots, A_N$,

the extended transition relation of $A_k, 1 \leq k \leq N$, denoted by $\looparrowright_{A_k}$, represents the transition relation of A_k together with self-loops on all states with events that are not in the alphabet of A_k

$$\looparrowright_{A_k} \triangleq \rightarrow_{A_k} \cup \{(q, \sigma, \acute{q}) \mid \forall q \in Q^A : \sigma \in (\Sigma^{A_1 \| \ldots \| A_N} \backslash \Sigma^{A_k}) \wedge \acute{q} = q\}.$$

By this extension, all automata in the DFA model have the same alphabet. This simplifies the full synchronous composition, since we only have to consider the first case. The extended explicit transition relation for an EFA can be similarly defined.

With the extended transition relation for each DFA defined, the characteristic function representing the transition relation of a set of DFAs can be computed as follows

$$\chi_{\rightarrow_{A_1 \| \ldots \| A_N}} = \bigwedge_{k=1}^{N} \chi_{\looparrowright_{A_k}}.$$

Regarding the computation for EFAs, it is not as straightforward as the DFA case, since the above formula doe not hold any more, i.e.,

$$\chi_{\mapsto_{E_1 \| \ldots \| E_N}} \neq \bigwedge_{k=1}^{N} \chi_{\looparrowright_{E_k}}.$$

Because then it would not be possible to keep track of the variables that are not updated. Furthermore, the action conflicts will disable the corresponding events. However, based on Definition 2.1.6, result should be a transition relations where values of variables are unchanged.

In [47], A monolithic approach to computing the characteristic function representing $\mapsto_{S_0}$ is presented. The approach, briefly described here, consists of three step:

1. Compute a characteristic function representing $\mapsto_{S_0}$ without including the actions, denoted by $\chi'_{\mapsto_{S_0}}$ and

   $$\chi'_{\mapsto_{S_0}} = \bigwedge_{k=1}^{N} \chi'_{\looparrowright_{E_k}},$$

   where $\chi'_{\looparrowright_{E_k}}$ denotes $\chi_{\looparrowright_{E_k}}$ excluding the update of variables.

2. Compute a characteristic function representing the update of the EFA variables, denoted by $\chi_{\mapsto^{v}_{S_0}}$.

   For two EFAs, E_1, E_2 and a variable v^i, we define

   $$\chi_{\looparrowright_{v^i, E_1 \| E_2}} = \bigvee_{j=1}^{4} (\chi_{C_j} \quad \chi_{\looparrowright_{v^i, E_1 \| E_2}}),$$

where $\chi_{\not\mapsto_{v^i,E_1\|E_2}}$ denotes the transition of $E_1 \parallel E_2$ where the variable v^i is updated. According to Definition 2.1.6, there are four alternative situations which are corresponded by

$$\bigvee_{j=1}^{4}(\chi_{C_j \not\mapsto_{v^i,E_1\|E_2}}),$$

can decide whether and how this variable v^i is updated. For the sake of simplicity and understandability,the formal definition is omitted and readers can find them in [47].

Finally, for a set of variables $v = (v^1, \ldots, v^n)$, $\chi_{\mapsto^v_{S_0}}$ can be computed as follows

$$\chi_{\mapsto^v_{S_0}} = \bigwedge_{i=1}^{n} \chi_{v^i,S_0}.$$

3. Based on $\chi'_{\mapsto_{S_0}}$ and $\chi_{\mapsto^v_{S_0}}$,

$$\chi_{\mapsto_{S_0}} = \chi'_{\mapsto_{S_0}} \wedge \chi_{\mapsto^v_{S_0}}.$$

The proof can be found in [47].

3.2.2 The Partitioning Approach

Another symbolic approach to the characteristic function representing $\mapsto_{S_0}$, is to apply partitioning techniques. Instead of obtaining one characteristic function, $\chi_{\mapsto_{S_0}}$, the partitioning approach generates a series of characteristic functions. These functions, with some logic connection in between, together represent $\chi_{\mapsto_{S_0}}$:

$$\chi_{\mapsto_{S_0}} = \chi^1_{\mapsto_{S_0}} \otimes \ldots \otimes \chi^m_{\mapsto_{S_0}},$$

where $\otimes$ denotes either conjunction or disjunction operator.

The monolithic approach works fine for some cases when applied to industrial applications. However, for other complex applications, the symbolic computation still causes the state-space explosion problem, because of either of the following two reasons:

- For some larger and more complex systems, the monolithic BDD representing $\mapsto_{S_0}$ is too huge to be constructed, although it occurs rarely.
- In the safe-state algorithm, the final set of safe states is generated after a series of (co-)reachability computations. These (co-)reachability computations, for instance `restricted backward reachability computation`, are the bottleneck of the algorithm. Assuming that the monolithic BDD

representing $\mapsto_{S_0}$ is managed to be constructed, however, the reachability computations may still suffer from the state-space explosion due to the large intermediate BDDs.

To reach significant intermediate BDD reduction, it is crucial to explore the search space in an intelligent way. The key is to impose structure on the state-space exploration. Moreover, to realize such an intelligent state-space exploration, an important ingredient is the use of *partitioning techniques*, which was rigorously defined in [20] and used in a number of contexts [48, 49]. There also exists a number of papers dealing with the adaption of these techniques to the verification or synthesis task of SCT. In [21, 22, 40], a straightforward but non-trivial approach was proposed to represents the monolithic transition relation of a fully synchronized DES modeled by DFAs by a collection of partial transition relations. In this thesis, the disjunctive partitioning is the technique that is always used. In [50], an alternative symbolic way to partition DESs modeled as EFAs by using the disjunctive partitioning technique. In the sequel, similar to the discussion of the monolithic approach. The partitioning approach for DFAs will be first described and extended to the EFA case.

First we need to introduce a couple of definitions which are used in the approach.

Definition 3.2.2. [Exclusive Dependency Set [22]] Given a set of DFAs, $A_1, \ldots, A_N$, $N \geq 2$, the dependency set of $A_i \in \{A_1, \ldots, A_N\}$, denoted by $D^+(A_i)$, can be defined as

$$D^+(A_i) = \{A_j \mid 1 \leq j \leq N \wedge j \neq i \wedge \Sigma^{A_i} \cap \Sigma^{A_j} \neq \emptyset\}.$$

Definition 3.2.3. [Retained Transition Relation] For $N \geq 2$ DFAs, the retained transition relation of $A_i, 1 \leq i \leq N$, denoted by $\curvearrowright_{A_i}$, is defined as

$$\curvearrowright_{A_i} = \{(q, \cdot, \acute{q}) \mid \forall q, \acute{q} \in Q^{A_i} \colon q = \acute{q}\},$$

where $\cdot$ denote any event $\sigma \in \Sigma^{E_1} \cup \ldots \cup \Sigma^{E_N}$.

Based on Definition 3.2.2 and 3.2.3, the characteristic function representing the partial transition relation of A_i is computed as

$$\chi^{A_i}_{\rightarrow_{S_0}} = \bigwedge_{A_j \in D^+(A_i)} \chi_{\rightarrow_{A_j}} \wedge \bigwedge_{A_k \notin D^+(A_i)} \chi_{\curvearrowright_{A_k}}.$$

Furthermore,

$$\chi_{\rightarrow_{S_0}} = \bigvee_{i=1}^{N} \chi^{A_i}_{\rightarrow_{S_0}}.$$

With respect to the EFA case, a slightly different approach is taken. The approach, called event-based partitioning approach, partitions the transition relation under full synchronous composition according to the included events of the model. In all of the following computations, let $\mathsf{E} = \{E_1, \ldots, E_N\}$ where $N \geq 2$.

For each event $\sigma \in \Sigma^{\mathbf{E}}$ where $\mathbf{E} = E_1 \parallel \ldots \parallel E_N$, the corresponding disjunctive partial transition relation $\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}}}$ under full synchronous composition can be constructed in the following three steps:

1. Compute a characteristic function of $\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}$, denoted by $\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}}$ where $\mathbf{E}^\dagger = E_1^\dagger \parallel \ldots \parallel E_m^\dagger$ and $\sigma \in \Sigma^{E_1^\dagger} \cap \ldots \cap \Sigma^{E_m^\dagger}$.
2. Compute a characteristic function of $\overset{\sigma}{\mapsto}_{\mathbf{E}^\ddagger}$, denoted by $\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\ddagger}}$ where $\mathbf{E}^\ddagger = E_1^\ddagger \parallel \ldots \parallel E_{m'}^\ddagger$ and $\{E_1^\ddagger, \ldots, E_{m'}^\ddagger\} = \mathsf{E} \backslash \{E_1^\dagger, \ldots, E_m^\dagger\}$.
3. Based on $\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}}$ and $\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\ddagger}}$, compute $\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}}}$.

Regarding step 1, computing $\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}}$, two further steps need to be performed in advance:

- Compute $\chi'_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}}$, which denotes the characteristic function of $\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}$ excluding the action functions of EFA variables.

  $$\chi'_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}} = \bigwedge_{k=1}^{m} (\exists\, \acute{b}^{V^i} . \chi_{\overset{\sigma}{\mapsto}_{E_k^\dagger}}),$$

  and

  $$\chi_{\overset{\sigma}{\mapsto}_{E_k^\dagger}} = \chi_{\mapsto_{E_k^\dagger}} \wedge \chi_\sigma,$$

  where $\chi_{\mapsto_{E_k^\dagger}}$ denotes the characteristic function representing the explicit transition relation of E_k while χ_σ denotes the characteristic function representing the event σ.

- Compute $\chi_{\overset{\sigma\ v}{\mapsto}_{\mathbf{E}^\dagger}}$ denoting the update of EFA variables.

Similar to the monolithic approach, for two EFAs. E_1 and E_2 and a variable v^i, we defined

$$\chi_{\overset{\sigma}{\mapsto}_{v^i, E_1 \parallel E_2}} = \bigvee_{j=1}^{4} \chi_{C_j(\overset{\sigma}{\mapsto}_{v^i, E_1 \parallel E_2})},$$

where $\chi_{\overset{\sigma}{\mapsto}_{v^i, E_1 \parallel E_2}}$ is the transition relation of $E_1 \parallel E_2$ where v^i is updated on the occurrence of σ. These four alternatives according to Definition 2.1.6, represented by

$$\bigvee_{j=1}^{4} \chi_{C_j(\overset{\sigma}{\mapsto}_{v^i, E_1 \parallel E_2})},$$

can decide whether and how this variable is updated once the event σ occurs. For the formal definition, refer to [50]. Hence, for a set of variables $v = (v^1, \ldots, v^n)$, $\chi_{\mapsto^{v}_{\mathbf{E}^\dagger}}$ can be computed as follows

$$\chi_{\mapsto^{v}_{\mathbf{E}^\dagger}} = \bigwedge_{i=1}^{n} \chi_{\mapsto^{\sigma}_{v^i,\mathbf{E}^\dagger}}.$$

At this stage, we are done with step 1.

Remark. Recall from Definition 2.1.6, that if there exists an event σ, such that $\sigma \in \Sigma^{E_1} \backslash \Sigma^{E_2}$, on the occurrence of σ, E_2 would remain the previous location, i.e. $\forall \ell, \acute{\ell} \in L^{E_2}, \ell = \acute{\ell}$. On the other hand, the values of variables are updated according to the transitions labeled by σ in E_1.

Regarding the scond step,

$$\chi_{\mapsto^{\sigma}_{\mathbf{E}^\ddagger}} = \bigwedge_{k=1}^{m'} \chi_{\curvearrowright^{\sigma}_{E_k^\ddagger}},$$

and

$$\curvearrowright^{\sigma}_{E_k^\ddagger} = \{(\ell, \sigma, \acute{\ell}) \mid \forall \ell, \acute{\ell} \in L^{E_k^\ddagger} \wedge \sigma \notin \Sigma^{E_k^\ddagger} \colon \ell = \acute{\ell}\}.$$

At this moment, we have done the first two steps. Sequentially, computing the characteristic function of $\mapsto^{\sigma}_{\mathbf{E}}$, can be computed:

$$\chi_{\mapsto^{\sigma}_{\mathbf{E}}} = \chi_{\mapsto^{\sigma}_{\mathbf{E}^\dagger}} \wedge \chi_{\mapsto^{\sigma}_{\mathbf{E}^\ddagger}}$$

Finally, for the closed-loop system S_0,we have

$$\chi_{\mapsto_{S_0}} = \bigvee_{\sigma \in \Sigma^{S_0}} \chi_{\mapsto^{\sigma}_{S_0}}.$$

The proof can be found in [50].

3.2.3 Structural State-Space Exploration

Following the previous sections, in order to design successful BDD-based reachability algorithms for large-scaled systems, it is vital to traverse the state-space in a structural way. For this purpose, an efficient symbolic algorithm for the DFA model, i.e. the workset algorithm was proposed in [22] while the in-depth analysis and proof can be found in [21].

As mentioned before, the interaction between two EFAs is not only affected by the shared events, but also the update of EFA variables. For instance, after an occurrence of an event, the values of variables are updated. These updated

variables may lead to some guards of other transitions from other EFAs to be evaluated to be true, even though they are labeled by different events. When designing the algorithm, this issue should be taken into account. Otherwise, the algorithm might either explore the state-space in an incorrect way or is not an exhaustive exploration.

In this section, the symbolic algorithm proposed in [50] is mainly discussed. The algorithm uses the event-partitioning approach and works for both EFA and DFA models. For the sake of simplicity and consistency with the notations in [50], the algorithms are described explicitly (without the characteristic functions), but they are implemented symbolically.

Algorithm 7 shows the forward reachability computation without forbidden states. Other versions of (co-)reachability computations can be similarly defined. As Algorithm 7 shows, taking as input the initial state and the set of partial transition relations of which each corresponds to each event, the algorithm maintains a set of active partial transition relations, W_k. For each iteration, one partial transition relation is selected and a saturated reachability search (Algorithm 8) is performed on it. If more reachable states are found, the *event and variable dependent transition relation sets* of σ, defined as follows, are appended to the workset. The algorithm terminates as long as there is no transition relation in W_k.

Definition 3.2.4 (Event Dependent Transition Relation Set of σ)**.** For $N \geq 2$ EFAs, $\mathsf{E} = \{E_1, \ldots, E_N\}$, the event dependent transition relation sets of σ, denoted by $D^e(\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N})$ is defined as:

$$D^e(\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N}) = \{\overset{\sigma'}{\mapsto}_{E_1\|\ldots\|E_N} |\ \sigma' \in D^e(\sigma) \wedge \sigma' \neq \sigma\},$$

where

$$\begin{aligned} D^e(\sigma) =& \{\sigma' \mid \exists E_i \in \mathsf{E}, \ell, \acute{\ell}, \check{\ell} \in L^{E_i}, v, \acute{v}, \check{v} \in V \\ & \Rightarrow (\ell, v, \sigma, \acute{\ell}, \acute{v}) \in \mapsto_{E_i} \wedge (\acute{\ell}, \acute{v}, \sigma', \check{\ell}, \check{v}) \in \mapsto_{E_i}\}. \end{aligned}$$

Definition 3.2.5 (Variable Dependent Transition Relation Set of σ)**.** For $N \geq 2$ EFAs $E_1, \ldots, E_N$ and a n-tuple of variables $v^1, \ldots, v^n$, the variable dependent transition relation sets of σ, denoted by $D^v(\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N})$, is defined as:

$$D^v(\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N}) = \{\overset{\sigma'}{\mapsto}_{E_1\|\ldots\|E_N} |\ \sigma' \in D^v(\sigma) \wedge \sigma' \neq \sigma\},$$

where

$$\begin{aligned} D^v(\sigma) =& \{\sigma' \mid \exists (\ell, \sigma', g, a, \acute{\ell}) \in \rightarrow_{E_1\|\ldots\|E_N}, \forall \chi_{v^i} \in g \\ & \Rightarrow \exists (\ell, v, \sigma, \acute{\ell}, \acute{v}) \in \overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N} \wedge v^i \neq \acute{v}^i)\} \end{aligned}$$

Algorithm 7: Event-based Forward Reachability

Input: the initial state q_0: $(\ell_0^{E_1} \times \ldots \times \ell_0^{E_N} \times v_0)$
Input: the set of partial transition relations W_0:
$\{\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N} | \ \forall \sigma \in \Sigma^{E_1} \cup \ldots \cup \Sigma^{E_N}\}$
Output: the reachable state set: Q_{reach}
begin
1 $k := 0$;
2 $Q^0 := \{q_0\}$;
repeat
3 Pick and remove $\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N} \in W_k$;
4 $k := k + 1$;
5 $Q^k := Q^{k-1} \cup \texttt{Reachability}(Q^{k-1}, \overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N})$;
6 **if** $Q^k \neq Q^{k-1}$ **then**
$W_k = W_{k-1} \cup D^e(\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N}) \cup D^v(\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N})$;
end
until $W_k = \emptyset$;
7 $Q_{reach} := Q^k$;
end

Algorithm 8: Reachability

Input: Q and $\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N}$
Output: the new reachable states found from $\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N}$: Q^{σ}_{reach}
begin
1 $l := 0$;
2 $Q^0 := Q$;
repeat
$l := l + 1$;
$Q^l := Q^{l-1} \cup \{(\acute{q}, \acute{v}) \mid \exists (q, v) \in Q^{l-1}$
$\Rightarrow (q, v, \sigma, \acute{q}, \acute{v}) \in \overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N}\}$;
until $Q^l = Q^{l-1}$;
$Q^{\sigma}_{reach} := Q^l$;
end

The correctness of the algorithm is proved in [50]. It should be mentioned that a drawback of the aforementioned event-based algorithm is that, more iterations are needed to reach the fixed point in the reachability task, especially for large-scaled systems involving a large number of events in the alphabet. To make such reachability computations more efficient, thus reducing the number of iterations, it is possible to combine multiple partial transition relations. This idea is originally presented in the literature [33], referring to as *clustering.* Based on this general principle, [50] discusses three invariants of this algorithms by using different ways to cluster BDDs together.

Chapter 4

User Cases

All the algorithms discussed from previous chapters have been implemented and integrated into Supremica [24, 25], which is a software tool for automatic verification, synthesis and simulation of DESs. Supremica is a project implemented in the Java programming language that was initiated about 10 years ago in the Automation research group at Chalmers University of Technology, and has been continuously developed since then by different PhD students.

This chapter is consisted of two parts. Firstly, an approach to modeling sequential resource allocation systems (RAS) by using EFAs, is presented and discussed. By modeling a set of simple RAS examples, we demonstrate how to use EFAs to model RAS with multiple instance execution, routing flexibility and failure handing. Moreover, two RAS benchmark examples are modeled and represented as EFAs. Secondly, the symbolic synthesis algorithm is applied to these two RAS examples, as well as other well-known DFA and EFA examples to generate the optimal supervisors.

4.1 Modeling Sequential Resource Allocation Systems using EFAs

For sequential resource allocation systems, we mainly focus on Conjunctive / Disjunctive sequential resource allocation systems (C/D RAS), one of the most powerful RAS classes investigated in the literature. In [51], a modeling approach was presented to model C/D RAS using EFAs. The section mainly demonstrates this approach by a set of simple examples.

C/D Resource Allocation Systems (RAS)

Definition 4.1.1. A *Conjunctive / Disjunctive resource allocation system* is formally defined by a 4-tuple [52, 53]:

$$\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$$

where

- $\mathcal{R} = \{R_1, \ldots, R_m\}$ is the set of the system *resource types*;
- $C : \mathcal{R} \rightarrow Z^+ -$ is the *capacity* function, characterizing the number of identical units from each resource type available in the system. Resources are assumed to be *reusable*, i.e., each allocation cycle does not affect their functional status or subsequent availability, and therefore, $C(R_i) \equiv C_i$ constitutes a system *invariant* for each i;
- $\mathcal{P} = \{\Pi_1, \ldots, \Pi_n\}$ denotes the set of the system *process types* supported by the considered system configuration. Each process type Π_j is a composite element itself, in particular, $\Pi_j = \langle \mathcal{S}_j, \mathcal{G}_j \rangle$, where $\mathcal{S}_j = \{\Xi_{j1}, \ldots, \Xi_{jl_j}\}$ denotes a set of *processing stages* involved in the definition of process type Π_j; $\mathcal{G}_j$ is an *acyclic diagraph* with the node set equal to $\mathcal{S}_j$. Every path in $\mathcal{G}_j$ connecting a ''source'' to a ''sink'' node corresponds to an execution sequence of Π_j;
- $\mathcal{A} : \bigcup_{j=1}^{n} \mathcal{S}_j \rightarrow \prod_{i=1}^{m} \{0, \ldots, C_i\}$ is the *resource allocation function* associating every processing stage Ξ_{jk} with the resource allocation vector $\mathcal{A}(\Xi_{jk}) \equiv A_{jk}$ required for its execution.

Furthermore, it is assumed that after a process instance accomplishes a non-terminal stage Ξ_{jk}, it must allocate the entire set of resources implied by the resource allocation request, in order to advance to the next stage. As soon as the requested resources are allocated, it releases all allocated resources that are not needed any more. The considered resource allocation protocol further guarantees that no resource type $R_i \in \mathcal{R}$ is over-allocated with respect to the capacity C_i at any processing stage.

Taking as input a RAS configuration, the approach in [51] can generate a set of extended finite automata, each of which models the resource allocation and deallocation of a process type. For simplicity and understandability, we start with a simple sequential RAS and first model it as a Petri net. For the readers who might be unfamiliar with Petri net, [3] provides a good introduction. From the Petri net model, the corresponding extended finite automata are then derived. After grasping the basic idea, extended finite automata are directly used to model the remaining C/D RAS.

Model Single-Unit (SU) RAS

Example 3. Consider a flexibly automated robotic cell example, borrowed from [52]. As Fig. 4.1 shows, the RAS is constituted by two process types Π_1 and Π_2, each of which consists of three processing stages performing the linear structure. The system resource set is $\mathcal{R} = \{R_1, R_2, R_3\}$, with the capacity $C_i = 1, i = 1, 2, 3$. Each processing stage $\Xi_{ij} (i = 1, 2; j = 1, 2, 3)$ requests one single unit of one resource type.

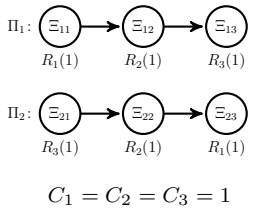


Figure 4.1: The considered SU RAS in Section 4.1

As the intermediate stage, the considered RAS is first modeled as a Petri net, shown in Fig. 4.2. For each resource type $R_i, i = 1, 2, 3$, the corresponding resource place is introduced. Initially the number of tokens of each resource place is set equal to its capacity. Similarly, three processing stage places for each process type are introduced to denote the number of process instances executing at the processing stages. For example, the stage places p_{11}, p_{12}, p_{13} map the three processing stages $\Xi_{11}, \Xi_{12}, \Xi_{13}$ of process type Π_1. Moreover, the transitions $t_{11}, t_{12}, \ldots, t_{23}$ depict the resource allocation and deallocation process. The weight of arcs from the resource places to transitions can be considered as the number of requested resource units with respect to various processing stages. From Fig. 4.2, it can be observed that multiple process instances can be allowed to execute in the Petri net model as long as the resource constraint is satisfied.

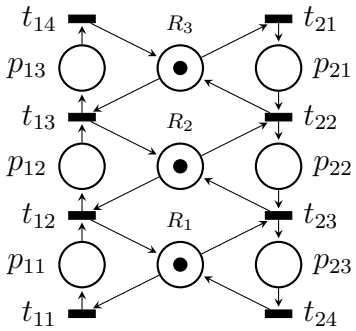
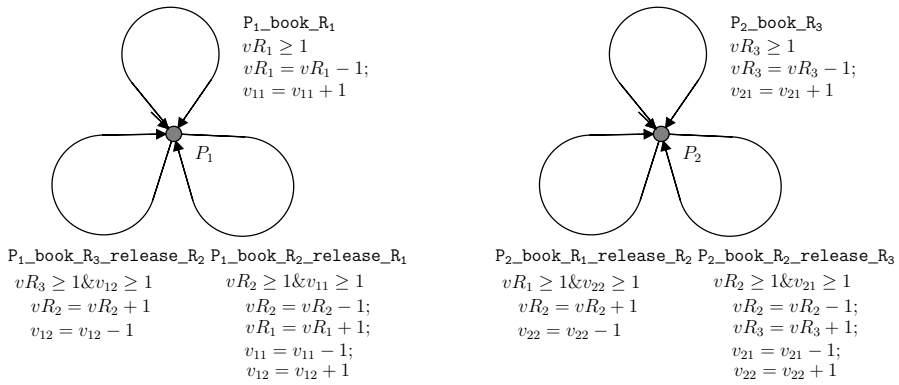


Figure 4.2: The Petri net model of the considered RAS in Section 4.1

With the considered RAS having been modeled as the Petri net, the extended finite automata can be correspondingly derived in the following steps:



(a) The EFA of Π_1 (b) The EFA of Π_2

Figure 4.3: EFAs modeling the two process types of Example 4.1.

- For each process type $\Pi_i, i = 1, 2$, create an extended finite automaton. To support the multiple instance execution as the Petri net does, each EFA is defined to have only one location and all the transitions labeled with events are added as self-loops. This location is both initial and marked.

- For each resource type (place in the Petri net) $R_i, i = 1, 2, 3$, declare one *resource variable* vR_i denoting the number of available units of R_i. The domain of vR_i is defined to be $\{0, \ldots, C_i\}$, where both of the initial and marked values of vR_i are equal to C_i.

- For each processing stage except the last one of each process type $\Pi_i, i = 1, 2$, declare one *instance variable* $v_{jk}, j = 1, 2$ and $k = 1, 2$, denoting the number of instances executing at the corresponding stage Ξ_{jk}. The domain for each instance variable is defined to be from 0 to the maximal number of executing instances. In this case, since each processing stage only acquires one unit of one resource type, the maximal number of instances at each processing stage is one. Therefore, the domain of all instance variables is defined to be $\{0, 1\}$ where 0 is the initial and marked value.

- Make use of the resource and instance variables defined above to construct the guards and actions. Guards are local formulae which determine whether a process instance can advance to the next processing stage while actions are used to update the available resource units and instances for various processing stages. Finally, the guards and actions are attached to the corresponding transitions of the created EFAs.

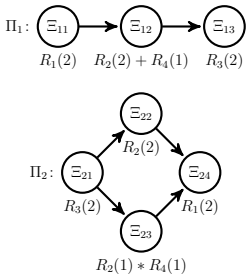
Fig. 4.3 shows the EFAs, which model the process type Π_1 and Π_2 based on the above steps. Here two points must be elaborated. From both the Petri net and the EFA, it can be observed that every time a process instance advances to the non-terminal processing stage, the requested resource allocation and the unused

resource deallocation occur simultaneously, which confirms to the assumption made in [53]. The purpose of the supervisor is to prevent the system from running into blocking situations. Since there is no restriction on these deallocation events as soon as the next requested resources are allocated, we know that all states that have resources waiting to be deallocated cannot be blocking states. Besides, it is noticed that there is no instance variable defined for the terminal processing stage. For the process instance at the terminal processing stage where the requested resources have been allocated, it is assumed that these allocated resources are released immediately. Certainly, this does not model the true behavior of the physical system, but enough information is captured. Reasonably, a model that can be used to find all blocking states need much less information than a model that expresses all possible events and variables, a important reduction of the system size is made.

Model C/D RAS

Compared with the Single-Unit (SU) RAS, e.g., the example shown above, modeling is more complicated in the context of the C/D RAS.

Example 4. Fig. 4.4 shows a C/D RAS which is extended from Example 4.1. The considered C/D RAS contains two process types Π_1 and Π_2. Same as before, the processing stages of Π_1 perform the linear structure, but the processing stage Ξ_{12} now allows for alternative resource type acquisition, i.e., either R_2 or R_4. The process type Π_2 is extended to have two alternatives to support the routing flexibility. The processing stage Ξ_{23} requires two types of resources to perform the task. Besides, the capacities of the resource types R_1, R_2, R_3 are increased to 4 and a new resource type R_4 with the capacity 2 is added into the C/D RAS.



$$C_1 = C_2 = C_3 = 4, C_4 = 2$$

Figure 4.4: The considered C/D RAS of Example 4

In order to model the process type Π_1 by following the previous instructions, the first issue we need to resolve is how to handle the processing stage Ξ_{12}. In particular, how to define the instance variables for it. The processing stage Ξ_{12}

allows for the alternative resource type acquisition. The decision that which resource type is allocated to an instance can only be determined dynamically. Besides, when an instance advances to the next stage Ξ_{13}, we cannot know which resource type should be deallocated. Therefore, two instance variables v_{12R_2} and v_{12R_4} need to be declared, which denote the number of instances having acquired R_2 and R_4 respectively. Based on the capacities of R_2 and R_4 and the requested units of each type, the domains of these two variables can be obtained, which are $\{0, \ldots, 4\}$ and $\{0, \ldots, 2\}$. With the instance variables defined, the corresponding EFA can be constructed, as Fig. 4.5 shown.

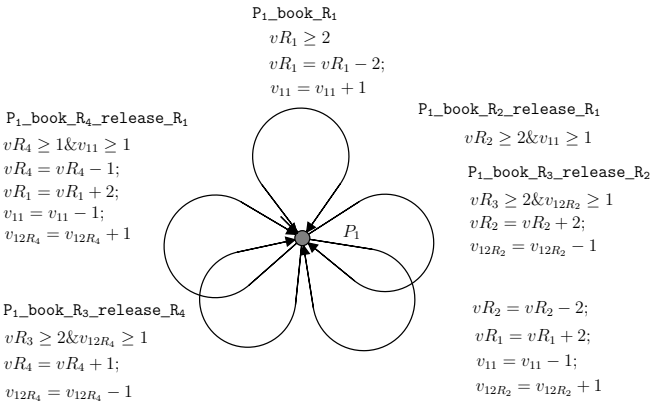


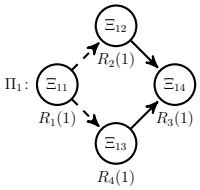
Figure 4.5: The EFA model of Π_1 of Example 4

With the experience of modeling the behavior of the alternative resource type acquisition, modeling the flexible routings follows the same strategy. Actually, the resultant EFA for Π_2 is similar to the EFA for Π_1, even though the processing stages perform different structures. Note that for the processing stage Ξ_{23}, one variable is enough to model the resource allocation and deallocation for this processing stage. The upper bound of the variable is defined to be the maximal number of instances executing at Ξ_{23} with both resource type units.

Model the Abnormal Behavior

The aforementioned modeling methods presume that the considered RAS is totally controllable. Specifically, (1) all the resource allocation events taking place can be disabled by the supervisor if necessary; (2) In a process type presenting routing flexibility, process instances can be conducted by the supervisor to choose different routing options to realize the system flexibility. However, in many contemporary applications, it is necessary to have some form of error handling. When an error occurs for an instance at some processing stage, repair or rework must be performed.

The idea to model such error handling is to introduce alternative branches after the necessary processing stage. Being different from modeling the routing



$$C_1 = C_2 = C_3 = C_4 = 1$$

Figure 4.6: The process type Π_1 with error handling

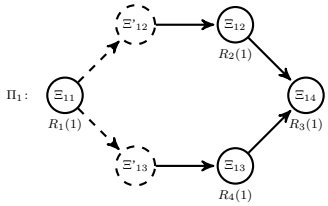


Figure 4.7: The process type Π_1 with imaginative stages

flexibility, the events corresponding to these uncontrollable branches are modeled as uncontrollable events. The supervisor cannot influence which branch to choose. Hereby, it must assure that there exists a non-blocking path for all branches.

Example 5. Consider the process type Π_1 of the example presented in Section 4.1. At this time, we suppose that an error may occur at the processing stage Ξ_{11} and needs to be handled by one unit of R_4. To distinguish from the flexible routing options, these two uncontrollable branches are described by dashed lines, as Fig. 4.6 shows.

As mentioned above, to model such alternatives, two uncontrollable events are introduced. Note that these two uncontrollable events have nothing to do with the resource allocation and deallocation. They are merely used to indicate the success and failure of an instance executing at Ξ_{11}. To assure that failed instances enter Ξ_{13} while successfully executed ones enter Ξ_{12}, two more EFA variables need to be declared. These two variables can be thought of as the variables of two imaginative stages, as Fig. 4.7 shows. An instance at either of these two imaginative stages still possesses the resources allocated to Ξ_{11}. Once the resource constraint is satisfied, it enters the next stage while the unused resources in Ξ_{11} are deallocated.

Fig. 4.8 shows the resultant EFA which models the process type Π_1 with error handling. Two uncontrollable events $!normal$ and $!abnormal$ indicate the success and failure of the instances executing in Ξ_{11}. Two variables iv_{12} and iv_{13} denote the number of instances which need enter Ξ_{12} and Ξ_{13} respectively. Note that it is only when an instance enters the next stage Ξ_{12} or Ξ_{13}, the resource (1 unit of R_1) is deallocated.

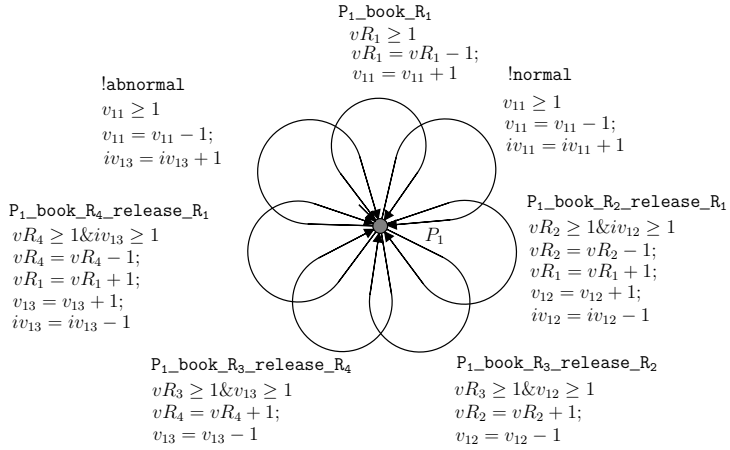


Figure 4.8: The EFA modeling error handling for Π_1

Two RAS Benchmark Examples

Following the above discussion, two RAS benchmark examples, as shown in Fig. 4.9 and Fig. 4.10, are modeled by using the modeling approach. The detailed exposition of these two benchmark examples can be found in [51]. Later the EFA models are fed into the symbolic algorithm to generate correct supervisors, i.e. deadlock-avoidance policies.

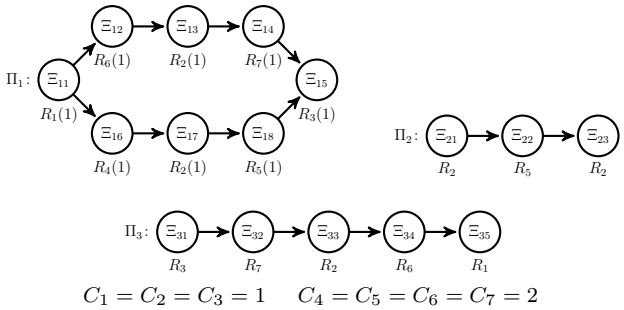


Figure 4.9: The flexible manufacturing system configuration (FMS) in [54].

4.2 Experimental Results

In this section, the efficiency of the proposed symbolic algorithm in this thesis is applied to a set of benchmark examples. Experiments are carried out on a standard PC (Intel Core 2 Quad CPU @ 2.4 GHz and 3GB RAM) running Windows 7 and the results and comparisons are shown in Table 4.1 and 4.2. The benchmark used to carry out the experiments are:

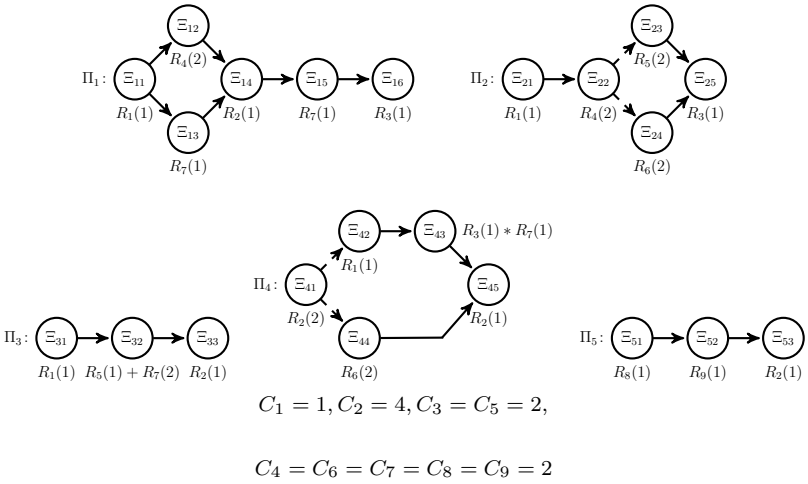


Figure 4.10: The extended D/C-RAS of Fig. 4.9 with error handling (RAS-EH).

- DFA models: Automated Guided Vehicles (AGV) [55]; Parallel Manufacturing Example [56]; The Transfer Line [57], Cat and Mouse [41] with different parameters.

- EFA models: Two RAS benchmark examples from the previous section, FMS and RAS-EH; Ball Sorting Process (BSP) [58]; Extended Cat and Mouse Tower (ECMT) and Extended Dinning Philosophers (EDP) [50] , with different parameters.

For DFA models, the comparison is made between two automat-based partitioning techniques: the conjunctive and disjunctive partitioning techniques of which both were implemented in Supremica. Table 4.1 shows the result of applying two partitioning techniques to the DFA examples explained above. It is observed that both of the partitioning based algorithms can handle the AGV and the Parallel Manufacturing example, for which the number of reachable states is up to 10^7. However, with DESs getting larger and more complicated, the conjunctive partitioning technique is not capable of synthesizing nonblocking and controllable supervisors and gets memory out (M.O.). The disjunctive partitioning, on the other hand, could successfully explore the state space within acceptable time. In addition, the column ‘‘BDD Peak’’, the maximal number of BDD nodes during the reachability computation, in the figure, shows that the disjunctive partitioning together with heuristic decisions can effectively reduce the number of intermediate BDD nodes.

For EFA models, the comparison is made between the symbolic monolithic and event-based partitioning. As Table 4.2 shows, for all of the benchmark examples, the maximal number of BDD nodes during the reachability computation of the partitioning synthesis approach is less than that in the monolithic synthesis

approach. Regarding the example Ball Sorting Process, event though the final number of supervisor states is only 706, the intermediate BDDs during the state-space exploration, on the other hand, are large due to the high interactive complexity of the system. The monolithic approach fails to explore the state-space while the partitioning approach can survive and synthesize the supervisor within 11 seconds. As mentioned before, since the partitioning algorithm is based on the alphabet which might contain a large number of events, more iterations than the standard algorithm are needed to reach the final fixed point. However, the intermediate BDDs produced during the computation are smaller, leading to improved memory and runtime efficiency. With respect to the last two benchmark examples, the Extended Cat and Mouse Tower and Extended Dining Philosophers, the partitioning approach can also handle some relatively large problem instances with the acceptable time while the monolithic algorithm is memory out or time out (>10 mins) However, with the values of parameters growing, both the computation time and memory used increase rapidly.

Table 4.1: Comparison Between Two Symbolic Partitioning Techniques (DFA)

			Conjunctive Synthesis		Disjunctive Synthesis	
Model	Reachable States	Supervisor states	BDD Peak (R)	Computation Time (s)	BDD Peak (R)	Computation Time (s)
AGV	22929408	1148928	9890	6.50	2850	0.87
Parallel Man	5702550	5702550	12363	2.47	2334	1.57
Transfer line (1,3)	64	28	17	0.05	13	0.10
Transfer line (5,3)	1.07×10^{9}	8.49×10^{4}	2352	1.69	299	0.59
Transfer line (10,3)	1.15×10^{18}	6.13×10^{13}	31022	48.36	1257	3.89
Transfer line (15,3)	1.23×10^{27}	4.42×10^{20}	M.O.	–	3032	12.80
CMT (1,1)	20	6	43	0.02	31	0.05
CMT (1,5)	605	579	2343	0.08	273	0.09
CMT (5,1)	1056	76	848	0.30	305	0.30
CMT (5,5)	6.91×10^{9}	3.15×10^{9}	M.O.	–	15964	20.86

Table 4.2: Comparison Between Two Symbolic Synthesis Approaches (EFA)

			Monolithic Synthesis		Disjunctive Synthesis	
Model	Reachable States	Supervisor states	BDD Peak (R)	Computation Time (s)	BDD Peak (R)	Computation Time (s)
RAS	1.19×10^4	0.88×10^4	2826	0.49	215	0.13
RAS-EH	1.84×10^6	0.68×10^6	42314	18.67	2275	0.87
BSP	706	706	M.O.	–	16640	10.48
ECMT (1,5)	605	579	447	0.01	255	0.02
ECMT (5,1)	1056	76	635	0.06	590	0.04
ECMT (1,7)	1198	1156	801	0.10	321	0.39
ECMT (7,1)	2710	155	1074	0.15	974	0.06
ECMT (3,3)	2.96×10^5	1.64×10^5	16770	24	5070	4.1
ECMT (5,5)	1.07×10^{10}	3.15×10^9	M.O.	–	65102	79
EDP (5,10)	167761	1596	1157	0.5	134	0.4
EDP (5,50)	3.46×10^8	1.38×10^5	7743	1.25	178	0.55
EDP (5,100)	1.05×10^{10}	1.05×10^6	–	T.O.	192	1.3
EDP (5,200)	3.28×10^{11}	8.20×10^6	–	T.O.	206	6.5

Chapter 5

Summary of Included Papers

Part II of this thesis consists of five papers. In this chapter the papers are summarized and important contributions are pointed out. It is also briefly discussed how the papers relate to each other.

Paper 1

> Zhennan Fei, Sajed Miremadi, Knut Åkesson, Bengt Lennartson. Symbolic State-Space Exploration and Guard Generation in Supervisory Control Theory. *will appear as a book chapter in volume 0271 of the Communications in Computer and Information Science (CCIS) series, 2012*.

The paper adapts a symbolic supervisory synthesis approach from prior work [22] to the guard generation procedure [23], making it applicable for industrially interesting applications. In particular, by using one of partitioning techniques, i.e., the disjunctive partitioning technique, the symbolic approach splits the state-space of discrete event systems under full synchronous composition into a set of simpler components and the reachability search is performed structurally. Moreover, the guard generation procedure is tailored to use the partitioned structure to extract the simplified guards and attach them to the original models. Finally, a comparison of algorithm efficiency between two partitioning techniques is made by applying them to a set of benchmark examples.

Paper 2

> Zhennan Fei, Knut Åkesson, Bengt Lennartson. Symbolic Reachability Computation using the Disjunctive Partitioning Technique in

Supervisory Control Theory. *Proceedings of the 2011 IEEE Conference on Robotics and Automation (ICRA 2011), pp. 4364-4369, 2011*.

With respect to the symbolic approach in Paper 1, the paper improves the state-space traversal algorithm where a set of forbidden states are involved to ensure the exhaustive exploration is always performed. The correctness of the modified traversal algorithm is formally proved as well.

Paper 3

Zhennan Fei, Sajed Miremadi, Knut Åkesson, Bengt Lennartson. Efficient Supervisory Synthesis to Large-Scale Discrete Event Systems Modeled as Extended Finite Automata. *Submitted to possible journal publication* .

In Paper 1 and Paper 2, the modeling formalism adopted to model DESs is deterministic finite automata (DFAs). This paper is intended to apply the disjunctive partitioning technique to DESs modeled as extended finite automata (EFAs) which are DFAs augmented with guards and actions. Due to the appearance of global variables, the corresponding full synchronous composition of EFAs is more complicated. This paper suggests an alternative way to construct partial transition relations, called event-based partitioning approach. In addition, the correctness is formally proved. Moreover, similar to the efficient state-space exploration algorithm from Paper 1 and 2, this paper proposes a straightforward algorithm including the proof of correctness to realize the structural state-space exploration.

Paper 4

Zhennan Fei, Sajed Miremadi, Knut Åkesson: Modeling Sequential Resource Allocation Systems using Extended Finite Automata, *Proceedings of the Seventh Annual IEEE Conference on Automation Science and Engineering (CASE 2011), pp. 444-449, 2012*.

This paper might be considered as an application of the symbolic method in Paper 3. It presents an approach to modeling sequential conjunctive / disjunctive resource allocation systems (C/D RAS) by using EFAs. The proposed approach allows for multiple products execution and resource allocation, routing flexibility and error handling. With the model of a considered C/D RAS obtained, the symbolic method is then utilized to synthesize the supervisor symbolically and generate the guards.

Chapter 6

Conclusions and Future Work

As one of main obstacles when it comes to analysis of large-scale discrete event systems, the state-space explosion problem has been well-studied and alleviated for decades in the model checking area. In short words, the problem results from the failure of explicit enumeration of synchronized models with huge state-space due to lack of time and memory. According to this, a well-known approach to the state-space explosion problem is to symbolically represent discrete event models and compute supervisor by using BDDs, compact and operation-efficient data structures for representing Boolean functions.

The main objective of this thesis is to develop efficient symbolic algorithms using BDDs in Supervisory Control Theory to allow verification and synthesis of modern industrial applications. Specifically, the disjunctive partitioning technique is applied to different modeling formalisms to partition the closed-loop system under full synchronous composition. Subsequently, efficient algorithms are suggested to explore the state-space in a structural way. Moreover, the thesis presents a modeling formalism together with a symbolic algorithm for modeling timed discrete event systems and synthesizing non-blocking supervisors. The proposed approaches have been implemented in the supervisory control tool Supremica and applied to a set of academic and industrial examples. Experimental results show that the method successfully compute supervisors and generate guards for a set of relatively large and complex industrial models.

As a supplement, the symbolic algorithms discussed in this thesis have been integrated to the modeling and synthesis framework from prior work. Overall, the whole framework provides the convenience for users to model systems and obtain control functions in the same model domain. All symbolic computations are performed efficiently by BDDs using the symbolic algorithms presented in this thesis, which are transparent and the only interface users deal with is the EFA framework.

There are three directions towards which we could extend and improve the work in future. It is believed that a sub-optimal but well-functioning BDD vari-

able ordering can still dramatically enhance the performance of the symbolic algorithm proposed in this paper and thus larger and more complicated systems can be handled. Moreover, it is possible to combine our symbolic approach with some sophisticated synthesis techniques, such as compositional techniques, to improve the efficiency of the synthesis task further. As mentioned before, there are some similarities between SCT and AI planning. We believe that it is possible to exploit promising techniques and approaches in planning and apply them to SCT.

References

[1] B. Cipra, ‘‘How number theory got the best of the pentium chip.’’ *Science*, vol. 267, no. 5195, p. 175, 1995.

[2] M. Dowson, ‘‘The Ariane 5 software failure,’’ *ACM SIGSOFT Software Engineering Notes*, vol. 22, no. 2, p. 84, 1997. [Online]. Available: http://portal.acm.org/citation.cfm?doid=251880.251992

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

[4] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 2000.

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

[6] 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.

[7] ------, ‘‘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.

[8] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 1st ed., ser. Series in Computer Science. Addison-Wesley, 1979.

[9] M. Sköldstam, K. Åkesson, and M. Fabian, ‘‘Modeling of discrete event systems using finite automata with variables,’’ *Decision and Control, 2007 46th IEEE Conference on*, pp. 3387--3392, 2007.

[10] F. Sharique, G. Gore, J. Richardsson, and M. Fabian, ‘‘Verification of a Novel Approach in Control of Flexible Manufacturing Cells,’’ in *FAIM 2004 Intl. Conference on Flexible Automation and Intelligent Manufacturing*, Toronto, Canada, Jul. 2004.

[11] M. R. Shoaei, B. Lennartson, and S. Miremadi, ‘‘Automatic generation of controllers for collision-free flexible manufacturing systems,’’ in *6th IEEE International Conference on Automation Science and Engineering*. IEEE, Aug. 2010, pp. 368--373.

[12] K. Åkesson and M. Fabian, ‘‘Implementing Supervisory Control for Chemical Batch Processes,’’ in *1999 Control Applications*, Hawai’i, USA, 1999, pp. 1272--1277.

[13] K. Åkesson, M. Fabian, and A. Vahidi, ‘‘Coordination of batches in flexible production,’’ in *2003 American Control Conference*, Chicago, USA, 2000, pp. 2735--2739.

[14] M. Fabian and R. Kumar, ‘‘Mutually nonblocking supervisory control of discrete event systems,’’ *Automatica*, vol. 36, no. 12, pp. 1863--1869, 2000.

[15] E. M. Clarke, D. E. Long, and K. L. McMillan, ‘‘Compositional model checking,’’ in *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*. Piscataway, NJ, USA: IEEE, 1989, pp. 353--362.

[16] 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, Oct. 2006.

[17] S. Mohajerani, R. Malik, and M. Fabian, ‘‘Nondeterminism avoidance in compositional synthesis of discrete event systems,’’ in *2011 IEEE International Conference on Automation Science and Engineering*. IEEE, 2011, pp. 19--24. [Online]. Available: http://dblp.uni-trier.de/db/conf/case/case2011.html#MohajeraniMF11

[18] S. B. Akers, ‘‘Binary Decision Diagrams,’’ *IEEE Transactions on Computers*, vol. 27, pp. 509--516, Jun. 1978.

[19] R. Bryant, ‘‘Graph-based algorithms for boolean function manipulation,’’ *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677--691, 1986.

[20] J. R. Burch, E. M. Clarke, and D. E. Long, ‘‘Symbolic Model Checking with Partitioned Transition Relations,’’ in *Proceedings of the International Conference on Very Large Scale Integration*, ser. IFIP Transactions, vol. A-1. North-Holland, 1991, pp. 49--58.

[21] 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*, Ann Arbor, MI, USA, Jul. 2006, pp. 288--293.

[22] 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.

[23] S. Miremadi, K. Å kesson, 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.

[24] 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*, Rhodos, Greece, 2003.

[25] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, ''Supremica---an integrated environment for verification, synthesis and simulation of discrete event systems,'' in *Proceedings of the 8th international Workshop on Discrete Event Systems, WODES'08*, Ann Arbor, MI, USA, 2006, pp. 384--385.

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

[27] S. J. Russell and Norvig, *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall, 2003.

[28] E. D. Sacerdoti, ''The nonlinear nature of plans,'' in *Proceedings of the 4th international joint conference on Artificial intelligence, IJCAI'75*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1975, pp. 206--214.

[29] S. Edelkamp and F. Reffel, ''Deterministic State Space Planning with BDDs,'' University of Freiburg, Tech. Rep., 1999.

[30] R. M. Jensen and M. M. Veloso, ''OBDD-based Universal Planning: Specifying and Solving Planning Problems for Synchronized Agents in Non-Deterministic Domains,'' in *in Non-Deterministic Domains, Lecture Notes in Computer Science*, 1999, pp. 213--248.

[31] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, ''Symbolic Model Checking: 10^{20} States and Beyond,'' in *Proceedings of the Fifth Annual IEEE Symposium on e Logic in Computer Science, 1990. LICS '90,*, Jun. 1990, pp. 428--439.

[32] K. L. McMillan, ‘‘Symbolic Model Checking: An approach to the state explosion problem,’’ Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1992.

[33] 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. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=275352

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

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

[36] H. Andersen, ‘‘An introduction to binary decision diagrams,’’ Department of Information Technology, Technical University of Denmark, Tech. Rep., 1999.

[37] 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, Jul. 2006, pp. 419--426.

[38] C. Ma and W. Wonham, ‘‘Nonblocking supervisory control of state tree structures,’’ *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 782--793, May 2006.

[39] Z. Fei, S. Miremadi, and K. Åkesson, ‘‘Efficient Symbolic Supervisory Synthesis and Guard Generation,’’ in *3rd International Conference on Agents and Artificial Intelligence*, Rome, Italy, 2011, pp. 106--115.

[40] Z. Fei, K. Åkesson, and B. Lennartson, ‘‘Symbolic Reachability Computation Using the Disjunctive Partitioning Technique in Supervisory Control Theory,’’ in *Control*, 2011, pp. 4364--4369.

[41] S. Miremadi, K. Åkesson, M. Fabian, A. Vahidi, and B. Lennartson, ‘‘Solving two supervisory control benchmark problems using Supremica,’’ in *9th International Workshop on Discrete Event Systems, 2008, WODES 08.*, May 2008, pp. 131--136.

[42] C. Ma and W. Wonham, ‘‘STSLib and its application to two benchmarks,’’ in *9th International Workshop on Discrete Event Systems, 2008, WODES’08.*, May 2008, pp. 119--124.

[43] C. E. Shannon, ‘‘A Mathematical Theory of Communication,’’ *The Bell System Technical Journal*, vol. 27, pp. 379--423,625--656, 1948.

[44] 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.

[45] A. Aziz, S. Tasiran, and R. K. Brayton, ‘‘BDD variable ordering for interacting finite state machines,’’ in *Proceedings of the 31st annual Design Automation Conference, DAC ’94*. New York, NY, USA: ACM, 1994, pp. 283--288.

[46] E. M. Clarke, K. L. Mcmillan, X. Zhao, M. Fujita, and J. Yang, ‘‘Spectral Transforms for Large Boolean Functions withApplications to Technology Mapping,’’ *Form. Methods Syst. Des.*, vol. 10, no. 2-3, pp. 137--148, 1997.

[47] S. Miremadi, B. Lennartson, and K. Å kesson, ‘‘A BDD-Based Approach for Modeling Plant and Supervisor by Extended Finite Automata,’’ *IEEE Transactions on Control Systems Technology*, vol. PP, no. 99, pp. 1--15, 2011.

[48] A. Geldenhuys, Jaco and Valmari, ‘‘Techniques for Smaller Intermediary BDDs,’’ in *12th International Conference on Concurrency Theory*, ser. Lecture Notes in Computer Science, M. Larsen, Kim and Nielsen, Ed., vol. 2154. Springer Berlin / Heidelberg, 2001, pp. 233--247.

[49] G. Cabodi, P. Camurati, L. Lavagno, and S. Quer, ‘‘Disjunctive Partitioning and Partial Iterative Squaring: An Effective Approach for Symbolic Traversal of Large Circuits,’’ in *Proceedings of the 34th Design Automation Conference*. ACM Press, 1997, pp. 728--733.

[50] Z. Fei, S. Miremadi, K. Å kesson, and B. Lennartson, ‘‘Efficient Supervisory Synthesis to Large-Scale Discrete Event Systems Modeled as Extended Finite Automata,’’ Chalmers University of Technology, Göteborg, Sweden, Tech. Rep., 2012.

[51] Z. Fei, S. Miremadi, and K. Åkesson, ‘‘Modeling Sequential Resource Allocation Systems using Extended Finite Automata,’’ in *7th Annual IEEE Conference on Automation Science and Engineering, CASE’11*, Trieste, 2011, pp. 444--449.

[52] S. Reveliotis, *Real-time management of resource allocation systems: A discrete event systems approach*. Springer, 2004.

[53] A. Nazeem and S. Reveliotis, ‘‘A practical approach to the design of maximally permissive liveness-enforcing supervisors for complex

resource allocation systems,'' in *6th IEEE Conference on Automation Science and Engineering (CASE)*, Toronto, Ontario, Canada, 2010, pp. 451--458. [Online]. Available: http://www.isye.gatech.edu/$\sim$spyros/publications/CASE-2010.pdf

[54] J. Ezpeleta, J. M. Colom, and J. Martinez, ''A Petri net based deadlock prevention policy for flexible manufacturing systems,'' *IEEE Transactions on Robotics and Automation*, vol. 11, no. 2, pp. 173--184, 1995. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=370500

[55] L. E. Holloway and B. H. Krogh, ''Synthesis of Feedback Control Logic for a Class of Controlled Petri Nets,'' *IEEE Transactions on Automatic Control*, vol. 35, no. 5, pp. 514--523, 1990.

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

[57] W. Wonham, ''Notes on Control of Discrete Event Systems,'' Electrical and Computer Engineering,Toronto, Toronto, Canada, Tech. Rep., 1999.

[58] G. Cengic, ''A Control Software Development Method Using IEC 61499 Function Blocks , Simulation and Formal Verification,'' *Development*, pp. 22--27, 2008.