View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Chalmers Publication Library

THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Symbolic Supervisory Control of Timed Discrete Event Systems

SAJED MIREMADI

Department of Signals and Systems
Automation Research Group
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2012

Symbolic Supervisory Control of Timed Discrete Event Systems
SAJED MIREMADI
ISBN 978-91-7385-765-9

© SAJED MIREMADI, 2012.

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 3446
ISSN 0346-718X

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

Typeset by the author using LaTeX.

Chalmers Reproservice
Gothenburg, Sweden 2012

To my family

Abstract

With the increasing complexity of computer systems, it is crucial to have efficient design of correct and well-functioning hardware and software systems. To this end, it is often desired to *control* the behavior of systems to possess some desired properties. A specific class of systems is called *discrete event systems (DES)*. DES deal with ‘discrete’ quantities, e.g., “number of robots in a manufacturing cell”, and their processes are driven by instantaneous ‘events’, e.g., “start of a machine”. In this thesis, the focus is on DES and an extension of such systems, which also considers the time points at which the events may occur, called *timed DES (TDES)*. Real-time applications such as communication networks, manufacturing facilities, or the execution of a computer program, can be considered into TDES.

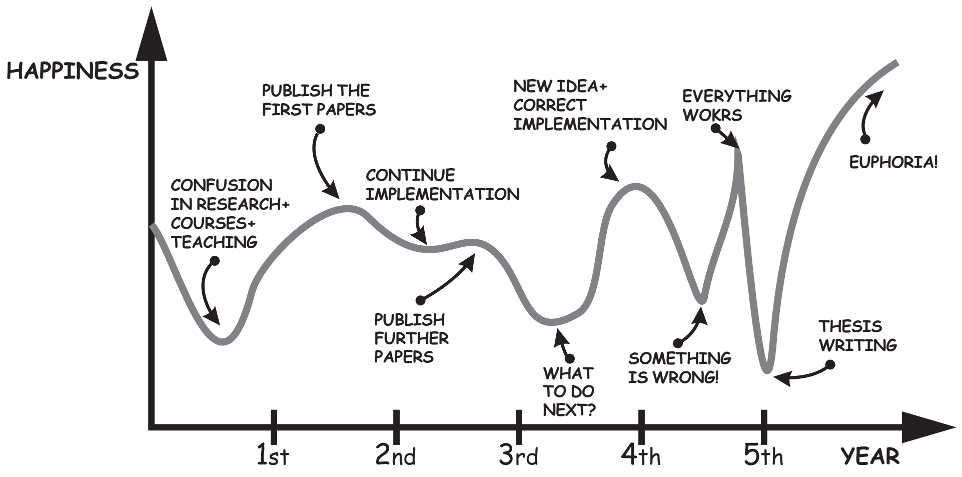
Having a DES or TDES, with some given *specifications*, by utilizing a well-known mathematical framework, called *supervisory control theory (SCT)*, it is possible to automatically generate a *supervisor* that restricts the system’s behavior towards the specifications, only when it is necessary. Applying the SCT to large and complex systems, typically follows with some issues, concerning computational complexity and modeling aspects, which is tackled in this thesis.

We model DES by *extended finite automata (EFAs)*, state transition models that contain discrete-valued variables. TDES are modeled by an augmentation of EFAs, called *timed EFAs (TEFAs)*, which contain a set of discrete-valued clocks. Based on EFAs or TEFAs, the supervisor can be *symbolically* computed, using *binary decision diagrams (BDDs)*, data structures that could, in many cases, lead to smaller representation of the state space. For complex systems, the computed supervisor may consist of many states, causing representation and implementation difficulties. To tackle this, based on the states of the supervisor, we symbolically compute logical constraints that will be attached to the original models to restrict the system’s behavior. Consequently, we present a framework, where given a set of EFAs or TEFAs, the supervisor is computed using BDDs, and represented in a modular manner based on the computed logical constraints. The framework has been developed, implemented, and applied to industrial case studies.

Keywords: Timed Discrete Event Systems, Supervisory Control Theory, Extended Finite Automata, Binary Decision Diagrams.

Acknowledgments

You start your PhD studies with the dream of making a major impact on the science! But soon you realize the reality is something different. More than contributing to the science, doing a PhD is about to learn how to ‘think’ in a structural and analytical manner. It is about to understand why you got correct results before getting happy, and why you got wrong results after becoming sad. Finally, it is about to write and formulate your results in a ‘convincible’ way, while meeting the ‘deadlines’. And during this journey, you indeed realize the power of procrastination! As a result, in five years, you deal with more or less happy moments, which can be summarized as below:



I would therefore like to thank the people that let me share my ‘peak’ moments with them, and cheered me up during the ‘troughs’. Initially, I want to thank my never-tiring supervisor Prof. Bengt Lennartson for supporting me in different aspects; and as the head of our research group, for treating it as his second family. And my co-supervisor Dr. Knut Åkesson for all the lively and fruitful discussions, which positively changed my way of thinking. I also would like to thank Prof. Martin “The Man in Black” Fabian for always being available for all kind of questions. All of my colleagues at the division of Automatic Control, Automation and Mechatronics really deserve a word of appreciation. Thank you guys, you are wonderful. A special appreciation goes to Zhennan “The Dude” Fei, for all the enjoyable discussions we had together and the unforgettable time we had in USA. Talking about USA, I would like to thank Prof. Spyros Reveliotis for giving us the opportunity to visit Georgia Tech. and experiencing the research environment at such a good university. Also, a special thank goes the administrative and technical staff at the department for always being so helpful and making everything work smoothly.

Finally, I would like to thank the family of Prof. Dadfar for their never-ending support, from the beginning of my studies in Sweden. My deepest gratitude goes to my family and friends, whom have always encouraged me and believed in me, especially, my parents and my brothers.

[illegible]

Sajed Miremadi
Gothenburg, November 2012

This work was carried out within the Wingquist Laboratory VINN Excellence Centre at Chalmers University of Technology and was also supported by Swedish Foundation for Strategic Research through the ProViking program.

Publications

This thesis is based on the following papers, included in full in Part II:

[Paper 1] S. Miremadi, K. Åkesson and B. Lennartson. Symbolic computation of reduced guards in supervisory control. *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 4, pp. 754-765, October 2011.

[Paper 2] S. Miremadi, B. Lennartson and K. Åkesson. A BDD-based approach for modeling plant and supervisor by extended finite automata. *IEEE Transactions on Control Systems Technology*, vol. 20, no. 6, pp. 1421-1435, November 2012.

[Paper 3] S. Miremadi, Z. Fei, K. Åkesson and B. Lennartson. Symbolic representation and computation of timed discrete event systems. Submitted to *IEEE Transactions on Automation Science and Engineering*, 2012.

[Paper 4] S. Miremadi, Z. Fei, K. Åkesson and B. Lennartson. Symbolic supervisory control of timed discrete event systems. Submitted to *IEEE Transactions on Control Systems Technology*, 2012.

The following papers are relevant to this work but not included in the thesis:

[1] S. Miremadi, Z. Fei, K. Åkesson and B. Lennartson. Symbolic computation of nonblocking control function for timed discrete event systems. To be published in *Proceedings of the* 8th *IEEE International Conference on Automation Science and Engineering*, December 2012.

[2] S. Miremadi and A. Voronov. Symbolic reduction of guards in supervisory control using genetic algorithms. Chalmers University of Technology, Gothenburg, Sweden, *Technical Report*, August 2012, p. 7.

[3] S. Miremadi, B. Lennartson and K. Åkesson. BDD-based supervisory control on extended finite automata. In *Proceedings of the* 7th *IEEE International Conference on Automation Science and Engineering*, August 2011, pp. 25-31.

[4] S. Miremadi, K. Åkesson and B. Lennartson. Extraction and representation of a supervisor Using guards in extended finite automata. In *Proceedings of the* 9th *International Workshop on Discrete Event Systems*, May 2008, pp. 193-199.

[5] S. Miremadi, K. Åkesson, M. Fabian, A. Vahidi and B. Lennartson. Solving two supervisory control benchmark problems using Supremica. In *Proceedings of the* 9th *International Workshop on Discrete Event Systems*, May 2008, pp. 131-136.

[6] Z. Fei, S. Miremadi, K. Åkesson and B. Lennartson. Efficient Supervisory Synthesis for Extended Finite Automata. Submitted to *IEEE Transactions on Control Systems Technology*, 2012.

[7] Z. Fei, S. Miremadi, K. Åkesson and B. Lennartson. Efficient supervisory synthesis to large-scale discrete event systems modeled as extended finite automata. In *Proceedings of the* 8th *IEEE International Conference on Automation Science and Engineering*, August 2012.

[8] Z. Fei, S. Miremadi, K. Åkesson and B. Lennartson. Modeling sequential resource allocation systems using extended finite automata. In *Proceedings of the* 7th *IEEE International Conference on Automation Science and Engineering*, August 2011, pp. 444-449.

[9] Z. Fei, S. Miremadi, K. Åkesson and B. Lennartson. Efficient symbolic supervisory synthesis and guard generation: Evaluating partitioning techniques for the state-space exploration. In *Proceedings of the* 3rd *International Conference on Agents and Artificial Intelligence*, January 2011, pp. 106-115.

[10] B. Lennartson, S. Miremadi, Z. Fei, M. Noori, M. Fabian and K. Åkesson. State-Vector Transition Model Applied to Supervisory Control. In *Proceedings of the* 17th *IEEE International Conference on Emerging Technologies and Factory Automation*, September 2012.

[11] M. Fabian, S. Miremadi, Z. Fei and K. Åkesson. Supervisory control of manufacturing systems using extended finite automata. To be published in *Formal Methods in Manufacturing* (Series on Industrial Information Technology), J. Campos, C. Seatzu and X. Xie, CRC Press/Taylor and Francis, 2013, ch. 10.

[12] M. R. Shoaei, S. Miremadi, K. Bengtsson and B. Lennartson. Reduced-order synthesis of operation sequences. In *Proceedings of the* 16th *IEEE International Conference on Emerging Technologies and Factory Automation*, September 2011, pp. 1-8.

[13] M. R. Shoaei, B. Lennartson and S. Miremadi. Automatic generation of controllers for collision-free flexible manufacturing systems. In *Proceedings of the* 6th *IEEE Conference on Automation Science and Engineering*, August 2010, pp. 368-373.

[14] K. Bengtsson, P. Bergagård, C. Thorstensson, B. Lennartson, K. Åkesson, C. Yuan, S. Miremadi and P. Falkman. Sequence planning using multiple and coordinated sequences of operations. *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 2, pp. 308-319, April 2012.

[15] K. Bengtsson, C. Thorstensson, B. Lennartson, K. Åkesson, C. Yuan, S. Miremadi and P. Falkman. Relations identification and visualization for sequence planning and automation design. In *Proceedings of the* 6th *IEEE Conference on Automation Science and Engineering*, August 2010, pp. 841-848.

Contents

List of Acronyms

BDD	–	Binary Decision Diagrams
CF	–	Characteristic Function
CS	–	Complement State
DES	–	Discrete Event System
DFA	–	Deterministic Finite Automaton
EFA	–	Extended Finite Automaton
EFSC	–	Extended Full Synchronous Composition
FA	–	Finite Automaton
FSC	–	Full Synchronous Composition
GA	–	Genetic Algorithms
IS	–	Independent State
PCG	–	Process Communication Graph
SCT	–	Supervisory Control Theory
STS	–	State Transition System
TA	–	Timed Automaton
TDES	–	Timed Discrete Event Systems
TEFA	–	Timed Extended Finite Automaton
TGA	–	Timed Game Automaton

Part I

Introductory Chapters

Chapter 1

Introduction

As we progress in time, the dependence and inseparability of our daily lives to hardware and software systems grow rapidly. For instance, modern cars, mobile phones, medical devices, communication systems, audio and video systems, control systems, etc. contain various types of software.

1.1 Discrete Event Systems

Historically, the systems that have been studied over the years involve quantities such as pressure, temperature, speed, and acceleration, which are continuous variables, evolving over time. Such systems have continuous states and are time-driven, i.e., a state changes as time changes. Since we can naturally define derivatives for continuous variables, modeling and analysis of such systems heavily rely on the theory and techniques related to differential and difference equations.

Nevertheless, not all system behaviors can be meaningfully represented by continuous variables and mathematical expressions. Most of the computer systems that we deal with include *discrete* properties. They are discrete in the sense that they are typically related to counting integer numbers such as the number of vehicles in a transportation system, number of faults in a system, or number of robots in a manufacturing cell. An interesting point about such systems is that most of them are driven by instantaneous *events* such as “start of a machine’ or “a traffic light turning green”. When an event occurs, the system transits from one *state* to another state, e.g., “the traffic light turns from amber to green”. A system which its state evolution depends entirely on the occurrence of asynchronous events over time is called a *discrete event system (DES)*[1], which is the scope of this thesis. Many systems are profitably modeled by DES such as manufacturing systems, operative systems, communication protocols and telephony systems.

[1]In the thesis, for ease of reading, “DES” is also used in plural form, i.e., “discrete event systems”.

In DES, merely the sequence of the visited states, i.e., the sequence that the events occur, is used to analyze different systems. In other words, the logical or the *qualitative* behavior of a system is in focus. For instance, in a manufacturing system a qualitative property could be "robot 1 should always complete its task before robot 2" or in a communication system "two users should not use a channel simultaneously". Nevertheless, the correct behavior of many real-time systems such as air traffic control systems and networked multimedia systems depends on the **delays** between events. In addition, in many cases, we also want to analyze the *quantitative* properties of the systems. For instance, in a manufacturing system we can check a property "if robot 1 does not finish its task in 20 seconds, let robot 2 finish its task" or in a communication system "if a channel is booked by a user for more than 1 minute, prohibit the user to use the channel and let another one use it". A DES that also considers the time points the events occur, is referred to as *timed DES (TDES)*. In this thesis, we analyze both DES and TDES.

With the increasing complexity of computer systems, it is crucial to have efficient design of correct and well-functioning hardware and software systems. Systems that do not work as expected can both lead to costly mistakes and disastrous consequences. In the early nineties, a bug was detected in Intel's Pentium II floating division unit, which caused the company a loss of about $475 million to replace faulty processors [1]. In 1997, the Mars Pathfinder landed on Mars, however, the spacecraft contained a design flaw that once in a while resulted in system resets and loss of important data [2]. Between 1985 and 1987, an error in the control part of the radiation therapy machine Therac-25 led to an overdose of radiation, which caused the death of six cancer patients [3]. All of these programs included *design* errors that were not captured during the design or implementation phases. Hence, somehow we need to ensure that the programs are correct or error-free, before putting them into practice.

1.2 Verification

As different systems are continuously used in larger contexts and in interaction with other components, they become more vulnerable to errors. It is known that the number of errors grows exponentially with the number of interacting system components. Thus, checking the correctness of complex systems with standard and conventional techniques such as random simulation or directed test are not always possible; especially, with the high demands on the system development time. Today, *formal verification* is mostly used for this purpose, that is mathematically-based techniques for proving or disproving the correctness of a property in a system [4, 5]. Investigations show that the design errors which were exposed in the aforementioned applications had been revealed if formal verification had been utilized. In formal verification, initially, the desired property to be verified is identified. Then, an abstract model of the system including the surrounding environment is built. Finally, the parts of the system that are interesting w.r.t the property are identified, and it is mathematically shown whether the property holds in the region of interest. Hence, the final result after verifying a system could be either *yes*, i.e., the system satisfies the given property, or *no*, i.e., the system does not satisfy the given property. Consequently, the goal is to design a *control function* that that restricts the system's behavior towards all the given desired properties.

1.3 Supervisory Control Theory

Basically, there are two conceivable ways of designing a control function: *manually* based on *verification* or *automatically* based on *synthesis*. In the verification method a control function candidate is designed manually in a fashion that supposedly controls the system in an appropriate manner. This is then verified towards some desired properties and if the result is satisfactory the control function design is finished. Preferably, the verification should give a hint about problems with the current control function so that the designer will have a better understanding of what needs to be changed. The verification method could be useful for applications, where **changes** are not applied frequently, e.g., microcontrollers. However, for applications, where the control function needs to be modified frequently due to changes to the system, the verification method could be quite time consuming. For instance, in a car manufacturing system, each time a new model is going to be produced, since much of the work is done on-line on the shop-floor, the production is down during the control function implementation. There are different tools such as UPPAAL [6] and KRONOS [7] that are based on the described verification procedure.

In the synthesis method, the above process is automated. Based on the specifications of the desired system behavior, synthesis generates a control function that makes sure the system does not violate the specifications. Naturally, synthesis can be carried out in different ways. For instance, it is possible to synthesize a control function that restricts the system more than necessary, which is typically not desired. In 1987, Ramadge and Wonham proposed a conceptual framework called *supervisory control theory (SCT)* for DES [8]. They showed that given a system, referred to as the *plant* and some desired properties, referred to as the *specifications*, there exists a control function, referred to as the *supervisor*, which is *minimally restrictive*. The supervisor is minimally restrictive in the sense that it restricts the plant only when it is necessary without violating the specifications. They also proposed a method to automatically synthesize such a supervisor. SCT has been applied to different domains such as manufacturing systems [9, 10], vehicular traffic [11], logistics [12], and communication networks [13, 14]. There are different tools such as Supremica [15] and TCT [16] that are based on the

SCT for generating control functions. In this thesis, we aim to compute control functions for DES and TDES, based on the SCT. Despite many benefits that can be gained by utilizing SCT, still, the control functions are mostly designed manually in the industry.

1.4 Challenges

In the following, we discuss some of the existing challenges in the SCT.

1.4.1 Supervisor Representation

The SCT is based on state-transition models; but industrial people are used to other representations such as sequential function charts (SFCs), ladder diagrams, Gantt charts, and PERT charts, that are exploited to represent the control functions. Specifically, the interpretation of a control function represented by a large and cluttered state-transition model requires the maintenance personnel to have other skills than are common today.

1.4.2 Qualitative and Quantitative Analysis

Conventional SCT is not defined for TDES. To this end, researchers proposed different approaches to, based on the SCT, perform *qualitative* analysis on TDES [17–20]. Most of these approaches are based on discrete time. There also exists many models and implementations that are suitable for *quantitative* analysis, most of them based on continuous time [21–25]; yet there are few works considering both the qualitative and quantitative aspects of TDES.

1.4.3 Computational Complexity

The complexity of a system represented by a state-transition model is often measured by its number of states, referred to as *state space*. The state space of a system grows exponentially by the addition of new components to the system. Since most of the industrial systems consist of many components, they include a huge state space, sometimes 10^{100} states or even more. Obviously, representing and enumerating such state spaces *explicitly* is more or less impossible both in terms of time and memory. To tackle this problem, the state space can be represented *symbolically (implicitly)*, which in many cases results in a smaller representation of the state space. Symbolic representation implies that the state space is described by means of logic constraints and special data structures, which makes it possible to simultaneously perform operations on a set of states, rather than a single state. One such powerful data structure is called *binary decision diagram (BDD)* that is used to symbolically represent Boolean functions [26]. It has been shown that BDD-based algorithms can improve the efficiency of computing control functions dramatically. For instance, in [27] the supervisor of a system with more than 10^{200} states was computed in a few minutes. However, in many cases it is quite complicated to represent models by BDDs and perform all the computations **purely** on these data structures, especially, with the introduction of time.

1.5 Contributions

The aforementioned challenges have been tackled in this thesis, which has lead to the following contributions:

C1: Symbolic representation of *extended finite automata (EFAs)*, finite automata extended with discrete variables, and their full synchronous composition operator, based on BDDs.

C2: Symbolic representation of *timed extended finite automata (TEFAs)*, EFAs extended with discrete-values clocks, and their full synchronous composition operator, based on BDDs. This contribution mainly considers the symbolic representation of time without including $tick$ events.

C3: Symbolic computation of the supervisor of TDES, modeled by TEFAs, based on BDDs.

C4: Identification of a subset of the states belonging to the supervisor as the *basic state sets*. Based on the basic state sets, some logical conditions, referred to as *guards*, are automatically generated. The guards express under which conditions an event is allowed to occur to fulfill the specifications.

C5: Symbolic computation of the basic state sets, using BDDs; and simplification of the guards, by utilizing the structure of the model and applying different heuristic techniques.

C6: Representation of a *modular* supervisor for a system that is modeled by TEFAs. The supervisor is modular in the sense that it is represented by the original TEFAs restricted by the computed guards.

C7: All algorithms are developed, implemented, and verified in Supremica [15, 28–30], a software tool for automatic verification, synthesis and simulation of DES.

In Table 1.1, the relationship between the main contributions and each of the mentioned challenges, i.e., supervisor representation (SR), qualitative and quantitative analysis (QQA), and computational complexity (CC), is illustrated. Further, the table shows in which appended papers the challenges are addressed and the contributions are presented.

Table 1.1: Illustration of the relationships: challenges – main contributions – appended papers.

Contribution	Challenge: SR	Challenge: QQA	Challenge: CC
C1			Paper 2
C2		Paper 3	Paper 3
C3		Paper 4	Paper 4
C4	Paper 1		
C5	Paper 1		Paper 1
C6	Paper 2		
C7			Paper 1-4

1.6 Outline

The thesis is divided in two parts. Part I provides introductory chapters that present background and context of the appended papers in Part II. The papers in Part II constitute the base of this thesis. A list of references is included at the end of Part I and at the end of each paper presented in Part II. All the proofs of the propositions, lemmas, and theorems in Part I are included in the appended papers in Part II.

Chapter 2 describes the modeling formalisms, deterministic finite automata and timed extended finite automata, which we used to model the systems. In Chapter 3, the supervisory control theory of both untimed discrete event systems and their timed extension are explained. Chapter 4 gives an overview of the symbolic data structures, i.e., binary decision diagrams, that are used to perform the analysis. Chapter 5 includes an illustrative and an industrial case study. A summary of the scientific papers, appended in Part II, is provided in Chapter 6. Finally, Part I is concluded in Chapter 7.

Chapter 2

Modeling Formalisms

When it comes to analysis and control of discrete event systems (DES), using appropriate modeling formalisms for representing the system's behavior is a dilemma. The appropriate choice highly depends on the objectives of the analysis. There are various modeling formalisms used to model DES such as finite automata [31, 32], Petri nets [33], process algebra [34, 35] and logic-based models [36].

Since automata are intuitive, easy to use, suitable for analysis and applicable to composition operations, they are used quite often for modeling, compared to other formalisms. In this work, automata are used to model DES. The main reason for this choice, is that automata conform well with supervisory control theory (discussed in Chapter 3), as they were used originally in [8]. In addition, to improve the expressiveness and compactness of the models, we use an extended variant of ordinary automata, where discrete-value variables and clocks are introduced to the model. In this work, we are interested in deterministic systems, and thus all models that are used in this work are considered to be deterministic.

Remark (SOS-notation)**.** A notation that will be used frequently is the *SOS-notation* (Structured Operational Semantics) [37]. The notation $\frac{\text{premise}}{\text{conclusion}}$ should be read as follows: if the proposition above the "solid line" (premise) holds, then the proposition under the fraction bar (conclusion) holds as well.

2.1 Finite Automata

A finite automaton (FA) is a state transition system or a state machine, formally defined as below.

Definition 2.1 Finite Automaton

A finite automaton (FA) is a 4-tuple $(Q, \Sigma, \mapsto, Q^0)$ *where*

- Q *is a finite set of states;*

- Σ *is a nonempty finite set of events;*
- $\mapsto \subseteq : Q \times \Sigma \times Q$ *is a transition relation; and*
- $Q^0 \subseteq Q$ *is a set of initial states.*

The set of events Σ is sometimes referred to as the *alphabet* of the automaton. The notation $|Q|$ denotes the number of states of the automaton. For an event σ, a *source-state* q and a *target-state* $\acute{q}$, a transition $(q, \sigma, \acute{q}) \in \mapsto$ is written $q \overset{\sigma}{\mapsto} \acute{q}$, which means that by the occurrence of σ, the system evolves from q to $\acute{q}$. A state q is said to be *reachable* if the automaton can evolve into q by a number of event executions, starting with an initial state.

Definition 2.2 Deterministic Finite Automaton (DFA)

An FA $(Q, \Sigma, \mapsto, Q^0)$ *is deterministic if there only exists a single initial state, i.e.,* $Q^0 = \{q^0\}$*; and*

$$\forall q \in Q : \ \frac{q \overset{\sigma}{\mapsto} \acute{q} \ \wedge \ q \overset{\sigma}{\mapsto} \grave{q}}{\acute{q} = \grave{q}}.$$

Informally, by executing an event at any state of a DFA, the next state can be determined. Hence, in a DFA, the transition relation will be a *function*. In the sequel, where ever we mention “automaton”, we refer to deterministic automaton.

For an automaton A, we use $\Gamma_A(q)$ to denote all the events in A that are *enabled* from state q. Formally, $\Gamma_A(q) = \{\sigma \in \Sigma \mid \exists \acute{q} \in Q_A : (q, \sigma, \acute{q}) \in \mapsto_A\}$. We also use the notation Q_A^σ to represent all the states in A, where event σ is enabled, i.e., $Q_A^\sigma = \{q \in Q_A | \sigma \in \Gamma_A(q)\}$.

It is often easier to model complex systems *modularly*, in a structured way, by a number of automata. The global behavior of a modular model can be represented by composing the automata. The composition of two automata is defined by the *full synchronous composition (FSC)* operator $\|$ [38]. In FSC, the shared events must be executed by all automata synchronously, while other events are executed independently.

Definition 2.3 Full Synchronous Composition (FSC)

For $k = 1, 2$*, consider two DFAs* $A_k = (Q_k, \Sigma_k, \mapsto_k, \{q_k^0\})$*. The* full synchronous composition (FSC) *of* A_1 *and* A_2*, denoted by* $A_1 \| A_2$*, is an automaton* $A = (Q, \Sigma, \mapsto, \{q^0\})$*, where*

- $Q = Q_1 \times Q_2$,
- $\Sigma = \Sigma_1 \cup \Sigma_2$,
- *the transition relation* $\mapsto \subseteq Q \times \Sigma \times Q$ *is defined based on the following rules:*

(a) $\sigma \in \Sigma_1 \cap \Sigma_2$:

$$\frac{(q_1, \sigma, \acute{q}_1) \in \mapsto_1 \;\; \wedge \;\; (q_2, \sigma, \acute{q}_2) \in \mapsto_2}{((q_1, q_2), \sigma, (\acute{q}_1, \acute{q}_2)) \in \mapsto},$$

(b) $\sigma \in \Sigma_1 \backslash \Sigma_2$:

$$\frac{(q_1, \sigma, \acute{q}_1) \in \mapsto_1 \;\; \wedge \;\; q_2 \in Q_2}{((q_1, q_2), \sigma, (\acute{q}_1, q_2)) \in \mapsto},$$

(c) $\sigma \in \Sigma_2 \backslash \Sigma_1$:

$$\frac{(q_2, \sigma, \acute{q}_2) \in \mapsto_2 \;\; \wedge \;\; q_1 \in Q_1}{((q_1, q_2), (q_1, \acute{q}_2)) \in \mapsto},$$

- $q^0 = (q_1^0, q_2^0)$.

In the above definition, $\Sigma_1 \backslash \Sigma_2$ denotes the set operation *relative complement*, indicating all the events that are included in Σ_1 but are not included in Σ_2. FSC can indeed be extended to multiple automata [38]. After the composition, the size of $A_1 \| A_2$, in the worst case, is the product of the sizes of A_1 and A_2. For the most, not all of these states are reachable–the size of $A_1 \| A_2$ can even be *smaller* than both A_1 and A_2–but the **growth** of the state-space can be considerable. This effect is particularly prominent when many automata are composed, in which the size of the state-space easily becomes unmanageable, a problem commonly referred to as the *state space explosion problem*. This is the problem that is tackled by representing the automata symbolically using binary decision diagrams, discussed in Chapter 4.

To show how a system can be modeled by FAs, let us take a look at an example, which is an extended version of the railroad example in [39].

EXAMPLE 2.1 *Railroad Crossing*

Consider a one-way railroad that crosses a one-way road, shown in Figure 2.1. It is desired to develop a control system that closes the gate when it receives a signal indicating that a train is approaching, and opens the gate when it receives a signal indicating that it has crossed the road and no other train has approached the crossing again. Furthermore, there exists a warning light on the road that has a reasonable distance to the crossing, indicating that a train is crossing the road to warn the drivers to slow down. The control system should only switch the light when the gate is closed and switch it off when the gate is opened. This

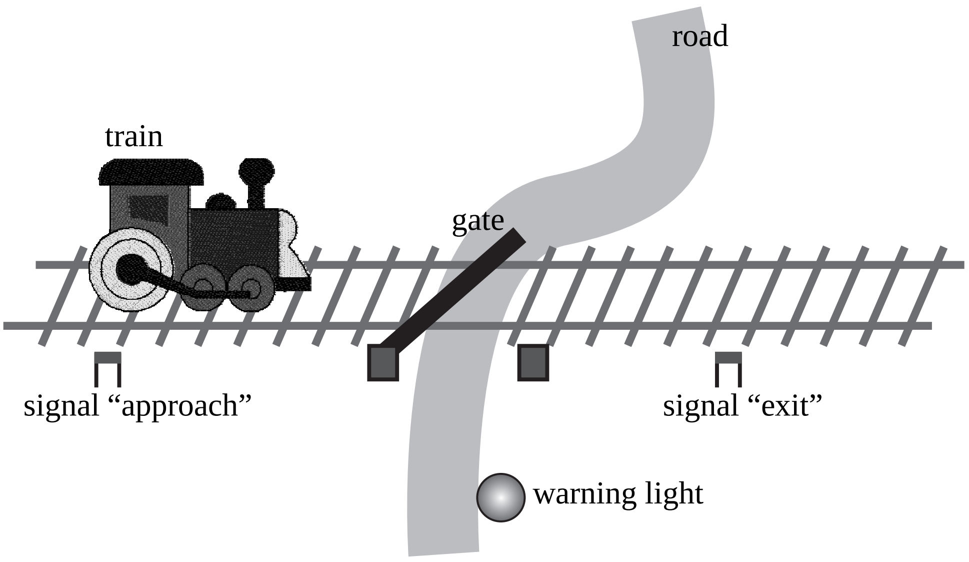


Figure 2.1: Railroad crossing example.

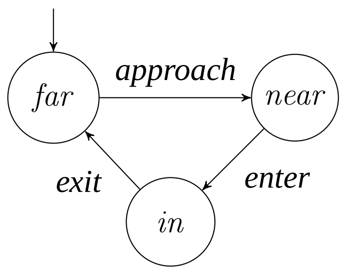
system can be modeled by four DFAs

$$\begin{aligned}\textbf{TRAIN} &= (\{far, near, in\}, \{\textit{approach}, \textit{enter}, \textit{exit}\}, \mapsto_1, \{far\}),\\ \textbf{GATE} &= (\{up, down\}, \{\textit{lower}, \textit{raise}\}, \mapsto_2, \{up\}),\\ \textbf{WARNINGLIGHT} &= (\{off, on\}, \{\textit{switch\_off}, \textit{switch\_on}\}, \mapsto_3, \{off\}),\\ \textbf{CONTROLLER} &= (\{l_0, \ldots, l_5\}, \{\textit{approach}, \textit{lower}, \textit{switch\_off},\\ &\qquad \textit{switch\_on}, \textit{exit}, \textit{raise}\}, \mapsto_4, \{0\}),\end{aligned}$$

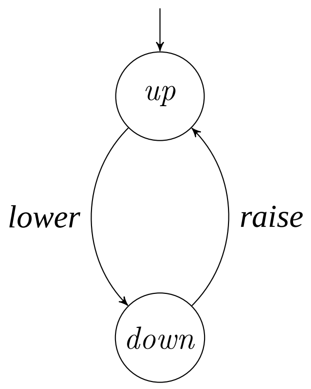
where their corresponding transition relations are depicted in Figure 2.2.

The states of the DFA representing the train (Figure 2.2a) have the following intuitive meaning: in state far the train is not close to the crossing, in state $near$ it is approaching the crossing and has just sent a signal to notify this, and in state in it is at the crossing. The states of **GATE** and **WARNINGLIGHT** have the obvious interpretation. The DFA **CONTROLLER** (Figure 2.2d) will evolve from state l_0 to l_1 when the event *approach* occurs. At state l_1, the controller closes the gate by sending the signal *lower* to the gate, ending up in state l_2, and turns the warning light on by sending the signal *switch_on* to the warning light, ending up in state l_3. When the event *exit* occurs, the train has left the crossing, ending up in state l_4. If at this moment, another train approaches the crossing, the controller will not open the gate and will evolve to state l_3; otherwise it opens the gate by sending the signal *raise* and turns off the warning light by sending the signal *switch_off*.

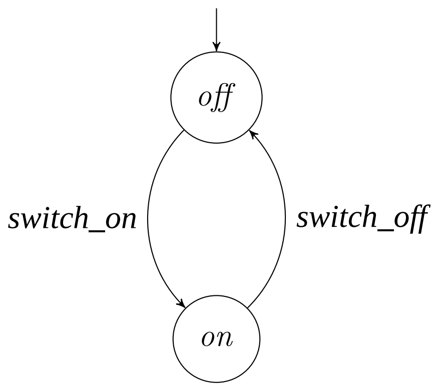
The global behavior of the system can be observed by synchronizing the automata: **TRAIN**$\|$**GATE**$\|$**WARNINGLIGHT**$\|$**CONTROLLER**. By considering the following two transitions in the synchronized DFA, it can be revealed



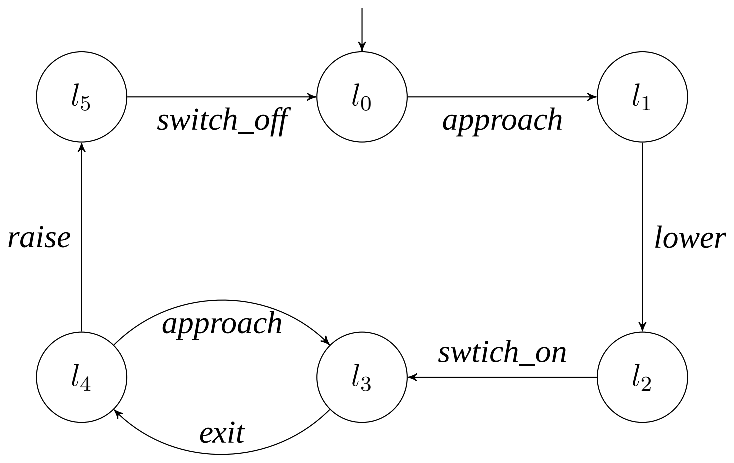
(a) TRAIN.



(b) GATE.



(c) WARNINGLIGHT.



(d) CONTROLLER.

Figure 2.2: DFAs modeling the railroad crossing example.

that the system suffers from a design flaw:

$$((\mathit{far}, \mathit{up}, \mathit{off}, l_0), \textit{approach}, (\mathit{near}, \mathit{up}, \mathit{off}, l_1)) \text{ and}$$
$$((\mathit{near}, \mathit{up}, \mathit{off}, l_1), \textit{enter}, (\mathit{in}, \mathit{up}, \mathit{off}, l_1)).$$

“At state $(\mathit{in}, \mathit{up}, \mathit{off}, l_1)$ the gate is about to close (by executing the event *lower*), while the train is (already) at the crossing, which can cause collision. In fact, the basic concept of the design is correct if and only if closing the gate does not take more time than the train needs to get to the crossing once it sends the signal *approach*” [39]. Such real-time constraints cannot be formulated by DFAs and will be the main motivation of introducing timed extended finite automata. □

2.2 Timed Extended Finite Automata

In some cases, modeling complex systems with DFAs can lead to incompact and intractable models for the users. One way to obtain more compact models is by introducing variables to the model. Naturally, physical signals that are stored in memories or sent between controllers can be modeled as global variables. For instance, a convenient way to model sensors and actuators is by using variables. Also, systems that have a buffer-resembling behavior can be easily modeled by variables. To this end, a new modeling formalism called *Extended Finite Automaton (EFA)*, was presented in [40]. An EFA is an augmentation of an FA with a finite set of discrete-valued variables. The variables appear in the transitions of the automata as either logical conditions, called *guards*, or updating function, called *actions*. A transition in an EFA is enabled if and only if its corresponding guard formula is satisfied; and when a transition is taken it may be followed by updates of variables defined by the associated actions. We model DES by using EFAs.

However, in order to model TDES, EFAs are not complete models to represent timing properties. To this end, we introduce *timed extended finite automaton (TEFA)*, which is an EFA, augmented with a finite set of discrete-valued clocks. Intuitively, a *clock* in a TEFA is a discrete variable in the sense of EFAs, restricted by some rules, mentioned later. The time implicitly elapses only at locations, whereas the transitions occur instantaneously with zero delay. It is worth to mention that by disregarding the clocks from TEFAs, the remaining formal discussions on TEFAs are equivalent to EFAs, and thus, in the following, we only discuss TEFAs.

Definition 2.4 Timed Extended Finite Automaton
A timed extended finite automaton is a 10-tuple

$$TE = (L, D^{\mathcal{V}}, \mathcal{C}, \Sigma, \rightarrow, Inv, L^0, D^{\mathcal{V}_0}, L^m, D^m),$$

where

- L *is a finite set of locations,*
- $D^{\mathcal{V}} = D_1^{\mathcal{V}} \times \ldots \times D_n^{\mathcal{V}}$ *is the domain of* n *variables* $\mathcal{V} = \{v_1, \ldots, v_n\}$*, where* $D_i^{\mathcal{V}}$ *is a finite set of integers,*
- $\mathcal{C}$ *is a finite set of* p *discrete valued clocks* $\{c_1, \ldots, c_p\}$*,*
- Σ *is a nonempty finite set of events,*
- $\rightarrow \subseteq L \times \mathcal{G}^{\mathcal{C}} \times \Sigma \times \mathcal{G} \times \mathcal{A} \times L$ *is the transition relation,*
- $Inv : L \rightarrow g^{\mathcal{C}}$*, is an invariant-assignment function,*
- $L^0 \subseteq L$ *is a set of initial locations,*
- $D^{\mathcal{V}_0} = D_1^{\mathcal{V}_0} \times \ldots \times D_n^{\mathcal{V}_0}$ *is a set of initial values of the variables,*
- $L^m \subseteq L$ *is a set of marked locations that are desired to be reached, and*
- $D^m = D^{\mathcal{V}_m} \times D^{\mathcal{C}_m}$ *is a set of pairs of marked valuations of the variables and clocks.*

In addition to $D^{\mathcal{V}}$, we also define $D^{\mathcal{C}}$ representing the domain of the p clocks. Later we will explain how the domain of a clock is defined and show that it is finite. The *global variable domain* denoted by $D_{\cup}^{\mathcal{V}}$ is the set that contains the values of all variables, defined formally as:

$$D_{\cup}^{\mathcal{V}} = \bigcup_{i=1}^{n} D_i^{\mathcal{V}}.$$

The *global clock domain* denoted by $D_{\cup}^{\mathcal{C}}$ is defined similarly. The largest value in $D_{\cup}^{\mathcal{V}}$ and $D_{\cup}^{\mathcal{C}}$ is denoted by $\mu\text{max}^{\mathcal{V}}$ and $\mu\text{max}^{\mathcal{C}}$, respectively. If a variable exceeds its domain, the result is not defined, and from an implementation point of view, it is upon the developer to decide how to implement such cases. For instance, the program can give the user a warning. In our implementation, values outside the domain will be ignored and will not be included in our computations. In contrast to variables, it is assumed that if a clock c_i reaches its maximum value, it will keep its value until it is reset. For a clock c_i, this behavior is modeled by a saturation function $\varrho_i : \mathbb{N} \rightarrow D_i^{\mathcal{C}}$:

$$\varrho_i(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \leq x < \mu\text{max}_i^{\mathcal{C}} \\ \mu\text{max}_i^{\mathcal{C}} & \text{if } x \geq \mu\text{max}_i^{\mathcal{C}} \end{cases},$$

where $\mathbb{N}$ is the set of natural numbers. The function $\varrho : \mathbb{N}^p \rightarrow D^{\mathcal{C}}$ is used to saturate the current value of all clocks.

The elements $\mathcal{G}$ and $\mathcal{A}$ are the sets of guards (conditional expressions) and action functions, respectively. In the TEFA framework, an arithmetic expression φ is formed according to the grammar

$$\varphi := \nu \mid v \mid c \mid (\varphi) \mid \varphi + \varphi \mid \varphi - \varphi \mid \varphi * \varphi \mid \varphi/\varphi \mid \varphi\%\varphi,$$

where $v \in \mathcal{V}$, $c \in \mathcal{C}$, $\nu \in D_{\cup}^{\mathcal{V}} \cup D_{\cup}^{\mathcal{C}}$, and $\%$ is the modulo operator. We use $\varphi^{\mathcal{V}}$ to denote an expression that does not contain any clocks and thus $\nu \in D_{\cup}^{\mathcal{V}}$. A *variable evaluation* for a variable $v_i \in \mathcal{V}$ is a function $\mu_i^{\mathcal{V}} : v_i \rightarrow D_i^{\mathcal{V}}$, assigning a value to the variable. A *clock evaluation* $\mu_i^{\mathcal{C}} : c_i \rightarrow D_i^{\mathcal{C}}$ is defined similarly. A set of evaluations for all variables and clocks is represented by $\mu^{\mathcal{V}}$ and $\mu^{\mathcal{C}}$, respectively.

A guard $g \in \mathcal{G}$ is a propositional expression formed according to the grammar

$$g \ := \ (g) \mid g^{\mathcal{V}} \wedge g^{\mathcal{C}} \mid g^{\mathcal{V}} \vee g^{\mathcal{C}},$$

where $g^{\mathcal{V}} \in \mathcal{G}^{\mathcal{V}}$ and $g^{\mathcal{C}} \in \mathcal{G}^{\mathcal{C}}$ are guards that are based on regular variables and clocks, respectively,

$$\begin{aligned} g^{\mathcal{V}} \ &:= \ \varphi^{\mathcal{V}} < \varphi^{\mathcal{V}} \mid \varphi^{\mathcal{V}} \leq \varphi^{\mathcal{V}} \mid \varphi^{\mathcal{V}} > \varphi^{\mathcal{V}} \mid \varphi^{\mathcal{V}} \geq \varphi^{\mathcal{V}} \mid \varphi^{\mathcal{V}} == \varphi^{\mathcal{V}} \mid \\ &\quad\ (g^{\mathcal{V}}) \mid g^{\mathcal{V}} \wedge g^{\mathcal{V}} \mid g^{\mathcal{V}} \vee g^{\mathcal{V}} \mid \top \mid \bot, \\ g^{\mathcal{C}} \ &:= \ c < \omega \mid c \leq \omega \mid c > \omega \mid c \geq \omega \mid c == \omega \mid (g^{\mathcal{C}}) \mid g^{\mathcal{C}} \wedge g^{\mathcal{C}} \mid \top \mid \bot, \end{aligned}$$

where $\top$ and $\bot$ represent Boolean logic `true` and `false`, respectively, and $\omega \in D_{\cup}^{\mathcal{C}}$. This implies that clocks can only be compared to constants. All nonzero values are considered as $\top$. The semantics of a guard g is specified by a *satisfaction relation* $\models$, indicating the pair of variable and clock evaluations $(\mu^{\mathcal{V}}, \mu^{\mathcal{C}})$ for which guard g is $\top$. It is written $(\mu^{\mathcal{V}}, \mu^{\mathcal{C}}) \models g$.

An action $\mathbf{a} \in \mathcal{A}$ is a tuple of functions:

$$\mathbf{a} = (\mathbf{a}^{\mathcal{V}}, \mathbf{a}^{\mathcal{C}}) = ((a_1^{\mathcal{V}}, \ldots, a_n^{\mathcal{V}}), (a_1^{\mathcal{C}}, \ldots, a_p^{\mathcal{C}})).$$

A variable action $a_i^{\mathcal{V}} : D^{\mathcal{V}} \times D^{\mathcal{C}} \rightarrow D_i^{\mathcal{V}}$ is a function that updates a variable; and a reset action $a_i^{\mathcal{C}} : D^{\mathcal{C}} \rightarrow 0$ is a function that only resets a clock. Hence, for a variable, the action is formed as $v_i = \varphi$ and for a clock it is formed as $c_i = 0$. An action function a_i that does not update a variable or clock is denoted by ξ, which is later used in the synchronization process to determine the updated value of v_i. Function Inv assigns to each location a *location invariant* that constrains the amount of time that may be spent in the location. Specifically, the location should be left before the invariant becomes invalid. Semantically, this situation causes time evolution to halt. Intuitively, if a location invariant consists of a *less than* relation, the invariant can be considered as a deadline.

The clocks can be seen as regular variables that are synchronized with a *global digital clock*. The clocks will evolve implicitly at the locations, each time the global clock "ticks". In other words, all clocks evolve synchronically at rate one. The value of a clock denotes the amount of time that has been elapsed since its last reset. Potentially, the clocks in $\mathcal{C}$ can have an infinite domain because the time will elapse forever. Nevertheless, based on the following argument a finite domain can be considered for each clock. Among the possible values of a clock, only a subset is relevant: those that can impact the guards' evaluations. For instance, for a guard $c_1 \leq 4$, the values above 4 will all have the same impact on the guard; thus the relevant values of c_1 is $\{0, \ldots, 5\}$. Considering $\mu\text{largest}_i^{\mathcal{C}}$ to be the largest constant in the model (including all guards), which the clock c_i is compared to, the domain of the clock c_i is $D_i^{\mathcal{C}} = \{0, 1, \ldots, \mu\text{largest}_i^{\mathcal{C}} + 1\}$. Thus, $\mu\text{max}_i^{\mathcal{C}} = \mu\text{largest}_i^{\mathcal{C}} + 1$. Consequently, the domain of the clocks $D^{\mathcal{C}} = D_1^{\mathcal{C}} \times \ldots \times D_p^{\mathcal{C}}$ will be finite.

For a variable v_i, $D_i^{\mathcal{V}_0}$ consists of the initial values of v_i. Since TEFAs are specifically designed to conform to the supervisory control theory (described in Chapter 3), it becomes natural to include a set of marked location and values in the tuple of definition of a TEFA. If the set of marked locations, evaluations of a variable or a clock is empty, then the entire domain is considered as marked.The *states* of a TEFA is defined as $Q \subseteq L \times D^{\mathcal{V}} \times D^{\mathcal{C}}$. The state for a location ℓ, variable evaluations $\mu^{\mathcal{V}}$, and clock evaluations $\mu^{\mathcal{C}}$ is represented as $\langle \ell, \mu^{\mathcal{V}}, \mu^{\mathcal{C}} \rangle$. Based on the states of a TEFA, a state transition system can be defined.

Definition 2.5 State Transition System of a TEFA

Let $TE = (L, D^{\mathcal{V}}, \mathcal{C}, \Sigma, \rightarrow, Inv, L^0, D^{\mathcal{V}_0}, L^m, D^m)$ be a TEFA. Its corresponding state transition system (STS)*, denoted by **STS**$(TE) = (Q, \Sigma, \mapsto, Q^0, Q^m)$, is a 5-tuple where*

- *$Q = L \times D^{\mathcal{V}} \times D^{\mathcal{C}}$ is a finite set of states,*
- *Σ is a set of events,*
- *$\mapsto \subseteq Q \times \Sigma \times Q$ is an* explicit state transition relation *defined by the following rule:*

$$\frac{(l, \sigma, g, \mathbf{a}, \acute{l}) \in \rightarrow \ \wedge \ (\mu^{\mathcal{V}}, \mu^{\mathcal{C}}) \models g \ \wedge \ (\mu^{\mathcal{V}}, \mu^{\mathcal{C}}) \models Inv(l)}{(\langle l, \mu^{\mathcal{V}}, \mu^{\mathcal{C}} \rangle, \sigma, \langle \acute{l}, \mathbf{a}^{\mathcal{V}}(\mu^{\mathcal{V}}, \mu^{\mathcal{C}}), \mathbf{a}^{\mathcal{C}}(\mu^{\mathcal{C}}) \rangle) \in \mapsto}; \tag{2.1}$$

- *$Q^0 = L^0 \times D^{\mathcal{V}_0} \times \mathbf{0}^p$ is a set of initial states ($\mathbf{0}^p$ is a p-tuple of zeros),*
- *$Q^m = L^m \times D^m$ is a set of marked states, i.e., the states that are desired to end up in.*

Indeed an STS is a FA with marked states. We deliberately use this new terminology to avoid confusions.

As mentioned earlier, we are only interested in deterministic systems.

Definition 2.6 Deterministic TEFA

A TEFA is deterministic if its corresponding STS is deterministic (based on Definition 2.2).

In the sequel, where ever we mention "TEFA", we refer to deterministic TEFA.

Remark (Nonzenoness)**.** We have omitted requirements on the definition necessary for executability. From every reachable state, the TEFA should admit the possibility of time to diverge. For example, the automaton should not enforce infinitely many events in a finite interval of time. A TEFA satisfying this operational requirement is called *non-zeno* [39].

Similar to DFAs, FSC can be defined for TEFAs, referred to as *extended FSC (EFSC)*. For a model with a number of TEFAs, we assume that the variables $\mathcal{V}$ and clocks $\mathcal{C}$ are all *global*, i.e., they are shared between the TEFAs, and that the clocks evolve synchronously with the same rate.

Definition 2.7 Extended Full Synchronous Composition

Consider the following two TEFAs

$$TE_k = (L_k, D^{\mathcal{V}}, \mathcal{C}, \Sigma_k, \rightarrow_k, Inv_k, L_k^0, D^{\mathcal{V}_0}, L_k^m, D^m),$$

for $k = 1, 2$. *The Extended Full Synchronous Composition (EFSC) of* TE_1 *and* TE_2*, denoted by* $TE_1 \| TE_2$*, is defined as*

$$TE_1 \| TE_2 = (L, D^{\mathcal{V}}, \mathcal{C}, \Sigma, \rightarrow, Inv, L^0, D^{\mathcal{V}_0}, L^m, D^m),$$

where

- $L = L_1 \times L_2$,
- $\Sigma = \Sigma_1 \cup \Sigma_2$;
- *the transition relation* $\rightarrow \subseteq L \times \mathcal{G}^{\mathcal{C}} \times \Sigma \times \mathcal{G} \times \mathcal{A} \times L$ *is defined as follows,*

$$\begin{aligned} \rightarrow = \{(l, \sigma, g, \mathbf{a}, \acute{l}) \mid \forall (l, \sigma, g, \hat{\mathbf{a}}, \acute{l}) \in \rightharpoonup :& \\ \forall i \in \{1, \ldots, |\mathcal{V}|\} :& \\ (\hat{a}_i = \xi \wedge a_i = v_i) \vee& \\ (\hat{a}_i \neq \xi \wedge a_i = \hat{a}_i)\},& \end{aligned} \tag{2.2}$$

where

(a) $\sigma \in \Sigma_1 \cap \Sigma_2$*:*

$$\frac{(l_1, \sigma, g_1, \mathbf{a_1}, \acute{l}_1) \in \rightarrow_1 \;\; \wedge \;\; (l_2, \sigma, g_2, \mathbf{a_2}, \acute{l}_2) \in \rightarrow_2}{((l_1, l_2), \sigma, g, \hat{\mathbf{a}}, (\acute{l}_1, \acute{l}_2)) \in \rightharpoonup}$$

such that,

* $g = g_1 \wedge g_2$*,*

* *For* $i = 1, \ldots, |\mathcal{V}|$*,*

$$\hat{a}_i^{\mathcal{V}} = \begin{cases} a_{1,i}^{\mathcal{V}} & \textit{if } a_{1,i}^{\mathcal{V}} = a_{2,i}^{\mathcal{V}} \\ a_{1,i}^{\mathcal{V}} & \textit{if } a_{2,i}^{\mathcal{V}} = \xi \\ a_{2,i}^{\mathcal{V}} & \textit{if } a_{1,i}^{\mathcal{V}} = \xi \\ \xi & \textit{otherwise} \end{cases},$$

where $a_{k,i}^{\mathcal{V}}$ *is the action function belonging to* $\rightarrow_k$*, updating the* i*-th variable, and* $\hat{\mathbf{a}}^{\mathcal{C}}$ *is defined exactly as* $\hat{\mathbf{a}}^{\mathcal{V}}$ *but on clocks,*

(b) $\sigma \in \Sigma_1 \backslash \Sigma_2$*:*

$$\frac{(l_1, \sigma, g_1, \mathbf{a_1}, \acute{l}_1) \in \rightarrow_1 \;\; \wedge \;\; l_2 \in L_2}{((l_1, l_2), \sigma, g_1, \mathbf{a_1}, (\acute{l}_1, l_2)) \in \rightharpoonup},$$

(c) $\sigma \in \Sigma_2 \backslash \Sigma_1$*:*

$$\frac{(l_2, \sigma, g_2, \mathbf{a_2}, \acute{l}_2) \in \rightarrow_2 \;\; \wedge \;\; l_1 \in L_1}{((l_1, l_2), \sigma, g_2, \mathbf{a_2}, (l_1, \acute{l}_2)) \in \rightharpoonup},$$

- $\forall (l_1, l_2) \in L : Inv(l_1, l_2) = Inv(l_1) \wedge Inv(l_2)$*,*

- $L^0 = L_1^0 \times L_2^0$*, and*

- $L^m = L_1^m \times L_2^m$*.*

Intuitively, in (2), an action function of form $\hat{a}_i = \xi$ indicates that variable v_i keeps its current value. Similar to the proof in [38], it can be proved that the EFSC operator is both commutative and associative and can be extended to multiple TEFAs. Note that, in the case of multiple TEFAs, the transition relation $\rightharpoonup$ in (2) refers to all TEFAs. In other words, $\rightharpoonup$ should first be computed for all TEFAs and then replace ξ with the current value. In the above definition, also observe that when the action functions of TE_1 and TE_2 explicitly try to update a shared variable to different values, we assume that the variable is not updated. It can indeed be discussed whether such a transition should be executed, nevertheless, such a situation is usually a consequence of bad modeling.

EXAMPLE 2.2 *Timed Railroad Crossing*

Recall Example 2.1, and the issue of not being able to specify real-time constraints. Let us assume that a train does not exceed a certain maximum speed. For each component, the following timing properties are considered:

TRAIN The train needs more than 2 minutes to reach the crossing after sending the *approach* signal; and it leaves the crossing 5 minutes after approaching it, at the latest.

GATE Lowering the gate takes at most 1 minute, and raising it takes at least 1 and at most 2 minutes.

CONTROLLER When the controller receives the signal *approach*, after exactly 1 minute it will close the gate by sending the signal *lower*. After receiving the *exit* signal, the controller raises the gate only if another train does not approach the crossing within 1 minute.

This timed system can be modeled by the following TEFAs

$$\begin{aligned}
\textit{Train} &= (\{far, near, in\}, \emptyset, \{c_1\}, \{\textit{approach}, \textit{enter}, \textit{exit}\}, \rightarrow_1, Inv_1,\\
&\qquad \{far\}, \emptyset, \{far\}, \emptyset),\\
\textit{Gate} &= (\{up, comingdown, down, goingup\}, \emptyset, \{c_2\},\\
&\qquad \{\textit{lower}, \textit{closed}, \textit{raise}, \textit{opened}\}, \rightarrow_2, Inv_2, \{up\}, \emptyset, \{up\}, \emptyset),\\
\textit{Controller} &= (\{l_0, \ldots, l_3\}, \{0, 1\}, \{c_3\}, \{\textit{approach}, \textit{lower}, \textit{exit}, \textit{raise}\}, \rightarrow_3,\\
&\qquad Inv_3, \{0\}, \{0\}, \{0\}, \{0\}),
\end{aligned}$$

where their corresponding transition relations and invariants are depicted in Figure 2.3. The invariants are illustrated by putting guards in the locations and a marked location is illustrated by a double line around the location. Compared to the DFA in Figure 2.2, it can be observed that the events *switch_off* and *switch_on* have been modeled by a variable $switch$ with domain {0,1}, where values 0 and 1 correspond to events *switch_off* and *switch_on*, respectively.

In the TEFA **GATE**, clock c_1 is set to zero on the occurrence of event *lower* and thus measures the elapse of time since that occurrence. Hence, the invariant $c_1 \leq 1$ at location $comingdown$ models the fact that the time delay between the occurrence of event *lower* and the change to location $down$ is at most 1 minute. Note that this would not have been established by putting a guard $c_1 \leq 1$ on the transition $(comingdown, \textit{closed}, down)$, as the value of c_1 would not refer to the time of occurrence *lower*. Similarly, the invariant $c_1 \leq 2$ at location $goingup$ indicates that raising the gate takes at most 2 minutes. No constraints are imposed on the residence time for locations up and $down$, i.e., $Inv_1(up) = Inv_1(down) = \top$.

In the TEFA **TRAIN**, on approaching the gate, clock c_2 is reset, and only if $c_2 > 2$ is the train allowed to enter the crossing.

The TEFA of the controller is depicted in Figure 2.3c and is forced to send the signal *lower* to the gate exactly after 1 minute after the train has signaled its approaching. In location l_3, the invariant $c_3 \leq 1$ indicates that if no other train comes within 1 minute, the signal *raise* should be sent to the gate.

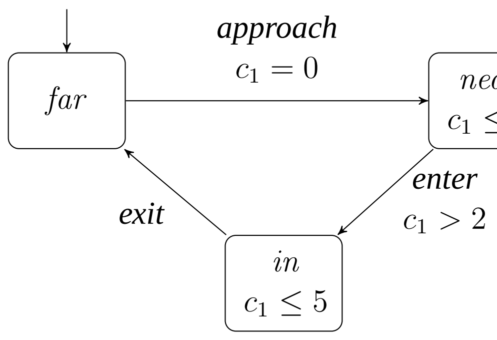
The synchronized TEFA **GATE**$\|$**TRAIN**$\|$**CONTROLLER** represents the global behavior of the system. From the definition of STS (2.5), the reachable states of the synchronized model is a subset of

$$\{\mathit{far}, \mathit{near}, \mathit{in}\} \times \{\mathit{up},\ \mathit{comingdown},\ \mathit{down},\ \mathit{goingup}\} \times \{l_0, \ldots, l_3\} \times \\ \{0, \ldots, 6\} \times \{0, \ldots, 3\} \times \{0, \ldots, 2\} \times \{0, 1\},$$

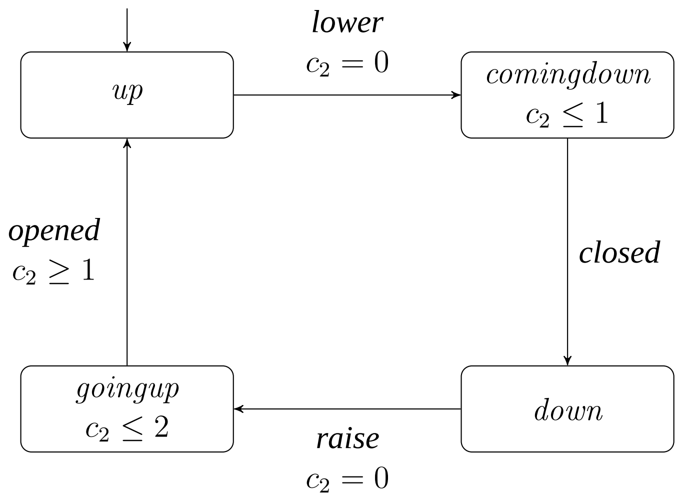
where $\{0, \ldots, 6\}$, $\{0, \ldots, 3\}$, $\{0, \ldots, 2\}$, and $\{0, 1\}$ correspond to the domains of c_1, c_2, c_3, and switch, respectively. Note that in the synchronized TEFA, the location $(\mathit{in}, \mathit{up}, l_1)$ is not reachable. In this location, the train is at the crossing while the gate is open. The location can only be reached when $c_1 > 2$, but as c_1 and c_3 are reset at the same time (on entrance of the preceding location), $c_1 > 2$ implies $c_3 > 2$, which is impossible due to l_1's invariant $c_3 \leq 1$. $\square$

2.3 Related Work

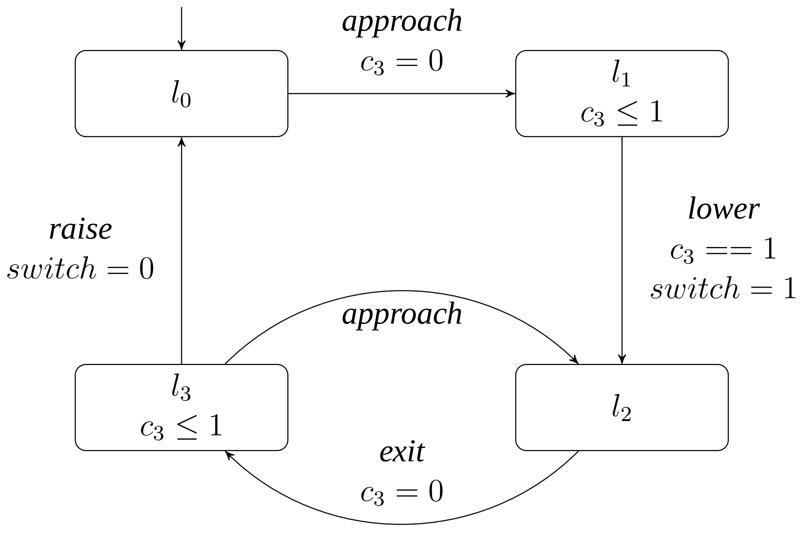
In model checking, a well-known modeling formalism that is used to model real-time applications, is *timed automata (TAs)* [21]. A timed automaton is a finite automaton extended with a finite set of real-valued clocks. Automated analysis of timed automata relies on the construction of a finite quotient of the infinite space of clock valuations. In an extended version, TAs can also include integer variables, denoted as ETAs [41]. Syntactically, TEFAs and ETAs are quite similar, however, from a semantical point of view, TEFAs are specifically designed to conform with the supervisory control theory. The main difference is how the composition operator is defined for TEFAs and TAs. In TEFAs, full synchronous composition is considered, where the synchronization is performed on all shared events and variables. In particular, two transitions can only be synchronized if both are labeled with the same shared event and if the guards are satisfied, while in TAs they also introduce a new type of events called *urgent channels* that can be taken as soon as they are enabled. Furthermore, the variable updates are treated differently. For a more elaborate and verbose exposition of TAs and their composition operator, refer to [41].



(a) TRAIN.



(b) GATE.



(c) CONTROLLER.

Figure 2.3: TEFAs modeling the timed railroad crossing example.

Chapter 3

Supervisory Control Theory

In 1987, Ramadge and Wonham showed that, for a DES, given a set models representing the behavior of the system, *plant*, and some desired properties, *specification*, there exists a unique control function, referred to as *supervisor*, that restricts the plant towards the specification, only when it is **necessary**. They called such a supervisor *minimally restrictive*. The main feature of a minimally restrictive supervisor is that it contains all the possible solutions a plant can be safely restricted towards the given specifications. This solution can later be used for quantitative analysis as well, such as time optimization. Later, they proposed a framework called *supervisory control theory (SCT)* [8], which is a mathematical framework for formal reasoning about supervision of systems modeled as DES. Traditionally, in SCT, a DES is based on formal languages, modeled by DFAs, and thus all the theory is defined on such models. In this chapter, in order to obtain compact models, we discuss how DES can also be modeled by EFAs. The supervisor will then be computed by transforming the EFAs to their corresponding FAs and applying conventional SCT.

However, the correct behavior of many real-time systems such as air traffic control systems and networked multimedia systems depends on the **delays** between events. Consequently, the researchers started to propose different approaches to apply SCT to TDES. There have been many attempts to model TDES and generalize SCT considering the real-time aspects [42]. These works can be divided into two categories; they are either based on *continuous* time or *discrete* time. In continuous time, the time is represented as real values while in discrete time, it is represented as integers. The question of which one to choose to model the systems is highly dependent on the structure of the specific applications and the properties that we want to check. For instance, in a manufacturing cell, where the components are synchronized by a PLC, discrete time is adequate to model the system and express most of its timing properties. A comparison between continuous and discrete time, according to their complexity and expressiveness, can be found in [43, 44]. In this thesis, we merely focus on discrete time.

The most settled framework, where SCT has been applied to TDES is a work carried out by Brandin and Wonham in 1994 [17], where a TDES is modeled by *timed transition models (TTMs)* [45]. In this framework, it is assumed that there exists a global digital clock. Furthermore, lower and upper time bounds are associated to the events to restrict their occurrence time points. To be able to apply the theory to TTMs, they transform such models to FAs by introducing a special event called $tick$, which represents the passage of time, and is generated by the global clock.

Similarly, in our framework, we model TDES by TEFAs; and in order to apply SCT, we transform the TEFAs to their corresponding EFAs by introducing a $tick$ event to the model. Note that in this manner, we do not need to directly define SCT for TEFAs and thus refer all the formal discussions about SCT on TEFAs to [17].

Finally, in this chapter, we discuss how the computed supervisor can be represented modularly by generating guards based on the states of the computed supervisor and attach them to the original models, in order to restrict their behaviors towards the specifications. Representing the supervisor modularly can be beneficial in cases where the supervisor consists of a large number od states.

3.1 SCT of Untimed DES

In this section, we describe the main concepts of SCT, defined for untimed DES. Figure 3.1 shows the *feedback loop* in the SCT. The plant spontaneously generates events in Σ that the supervisor can enable or disable as a function $f(\cdot)$ of the earlier behavior of the plant (the observed sequence of events). As assumed earlier, the plant is modeled by DFAs. In [46], it was shown that the FSC operator can be used to model the supervision. That is, the supervisor can be considered as an automaton too. For example, when a supervisor S supervises a plant P, the behavior that S tries to enforce is $P\|S$. Notably, if S is not designed properly, some parts of the plant may not be susceptible to the control imposed by S, so the actual behavior may be another. This is the reason why S should be synthesized using formal methods that guarantee that S does not try to control parts of the plant that can not be controlled or, in other words, that the *closed loop behavior* really is $P\|S$. In this work, we assume that the supervisor always *refines* the plant, that is, $S = P\|S$. We refer to the states of the supervisor as *safe states* and denote it by Q^{safe}.

The supervisor decides to enable or disable events based on a given specification in terms of an automaton. It is also possible to explicitly specify some states in the plant or the specification as *explicitly forbidden* states, that are states where the system should not end up in. As pointed out earlier, for real systems, modeling the plant or the specification as a single automaton may become very large and complex. Therefore, the plant and specification are typically modeled as a set of sub-plans $P_1, P_2, \ldots$ and sub-specifications $Sp_1, Sp_2, \ldots$, and thus the

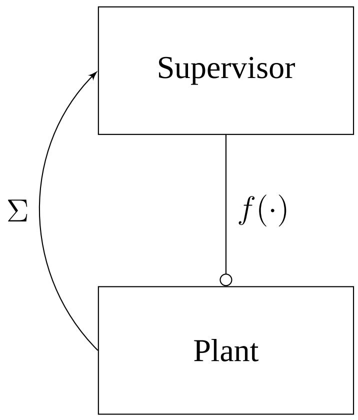


Figure 3.1: The feedback loop in the SCT.

plant and the specification will be represented by the composition of their sub-components, i.e., $P = P_1 \| P_2 \| \ldots$ and $Sp = Sp_1 \| Sp_2 \| \ldots$. For a composed automaton, a state is explicitly forbidden if at least one of its sub-states is explicitly forbidden in its corresponding automaton.

Controllability

In general, it is reasonable to assume that some events in the plant are not susceptible to disablement by a supervisor. For example, the plant may sometimes act randomly or have internal doings that the supervisor can have no influence on. To incorporate this, the SCT introduces the notions of *controllable* and *uncontrollable* events. Controllable events can be disabled by the supervisor while uncontrollable can not.

It is important that the supervisor is *controllable*, meaning that while it restricts the plant towards the specification, it never tries to disable uncontrollable events. To this end, the alphabet Σ of the plant is divided into two disjoint sets of controllable events Σ^c and uncontrollable events Σ^u. Controllability, is assumed to be universally defined, that is, if an event σ is controllable in one automaton it is controllable in all other automata that consider that event. In figures, uncontrollable events are prefixed by an exclamation mark "!".

The formal definition of controllability is defined as follows.

Definition 3.1 Controllability

Let G and K be two DFAs. A state $(p, q) \in Q_G \times Q_K$ is controllable *if,*

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

K is controllable with respect to G if, for every state (p, q) that is reachable in $G\|K$ it holds that (p, q) is controllable.

Intuitively, K is controllable with respect to G if, in any reachable state in the composition, the enabled uncontrollable events in G are also enabled in $G||K$. For the event to be enabled in $G||K$, it must not be disabled in the corresponding state of K. That is, the event must either be enabled in the current state of K or not even present in the alphabet of K, in which case that event can be thought of as enabled in all states of K.

Nonblocking

Even though a supervisor is controllable, it is not necessarily very useful. The supervisor guarantees that the plant does not violate the specification, however, the case may be that the supervisor restricts the plant from doing what it was supposed to do. For instance, the supervisor may allow the plant to get stuck somewhere, referred to as *deadlock*, or end up in a loop from which it can not get out, referred to as *livelock*. To care of this, states of particular interest in the plant and in the specification can be *marked*, denoted by Q^m. The idea, then, is to design the supervisor so that it always allows the plant to reach at least one of the states that both plant and the specification have marked. Such a supervisor is called *nonblocking*, which in SCT is a property that a supervisor should have.

In the following, the definition of the nonblocking property is given.

Definition 3.2 Nonblocking
Let G be a DFA. A state $q \in Q_G$ is said to be nonblocking *if, starting from q at least a marked state belonging to Q^m could be reached. G is nonblocking if, for every state q that is reachable, it holds that q is nonblocking.*

That is, an automaton is nonblocking when "all" reachable states can continue to reach some marked state. In a composed automaton, a state is marked if all its sub-states are marked in their corresponding automata. Essentially, the nonblocking states can be computed by taking the intersection between the reachable states and *coreachable states*, which are the states from which a marked state can be reached by a number of event executions.

Minimally Restrictiveness

A careful reader may have realized that there does not exist a unique controllable and nonblocking supervisor. It is possible to supervise one and the same system in many different ways. More specifically, it is possible to design a controllable and nonblocking supervisor that restricts the plant more than necessary. It is natural to regard a supervisor that restricts the plant at little as possible, referred to as a *minimally restrictive*[1] supervisor. Designing a minimally restrictive supervisor

[1]In some literature, it is also called maximally permissive, supremal, or optimal.

has several advantages. It gives the designers all the possible ways they can control a system, which could be beneficial from different perspectives. Especially, in this work, since we deal with timed systems, the supervisor will include some timing information, which can later be used for timing analysis. For instance, we may want to minimize the total time it takes to reach a marked state from any state in the supervisor. One way to this, is to have all possible solutions and select the proper ones.

In this thesis, we are interested in computing the **unique** controllable, nonblocking, and minimally restrictive supervisor, from now on, shortly “supervisor”.

3.1.1 DES Modeled by EFAs

So far, we have assumed that DES are modeled by FAs. It is also possible to model DES by EFAs that also include discrete-valued variables. The main benefit of using EFAs, as a modeling tool, is that the values of the variables in state transitions can be hidden, yielding compact models.

In the previous section, we explained the conventional SCT on DES modeled by FAs, where their transition relations are represented explicitly by their states and events. Hence, the theoretical framework of the conventional SCT cannot be directly applied to EFAs, where the states are implicitly represented in the models. The SCT can be applied to EFAs in two ways: 1) define a new theoretical framework for EFAs that conforms with the conventional SCT, or 2) transform the EFAs to their corresponding STS, i.e., FAs, and then apply the conventional SCT.

In [47], a theoretical framework is proposed, where SCT can be applied directly on EFAs. They symbolically compute the supervisor directly based on the EFAs by performing algebraic operations.

In this work, we follow the second approach, by transforming the EFAs to FAs having the same properties. In this way, by showing the correctness of the correlation between EFAs and FAs, the conventional SCT can be directly applied. Furthermore, FAs can easily be transformed to BDDs, described in Section 4.2.1, which are the symbolic representation used in this work. In Paper 2, it is shown how EFAs can be directly converted to BDDs, representing the corresponding FAs of the EFAs.

Transformation of EFAs to FAs

A single EFA can be directly transformed to FA by computing its corresponding STS, based on Definition 2.5. Given N EFAs $E_1, \ldots, E_N$, the global behavior of the system can be obtained by computing the corresponding STS of $E_1 \| \ldots \| E_N$. One could say why not transform each EFA to its corresponding STS and apply the FSC defined for FAs. However, in this way, the global behavior will not be the same:

$$\textbf{STS}(E_1\|\ldots\|E_N) \neq \textbf{STS}(E_1)\|\ldots\|\textbf{STS}(E_N). \tag{3.1}$$

This is because of the special treatment of the update of variables defined in the EFSC operator on EFAs (Definition 2.7). For instance, if a variable is not updated on a transition or if its action conflicts with an action on another transition, it is considered that the variable will keep its current value. Intuitively, the shared variables interact via the EFSC and can via their action functions exchange information during the synchronization process. To obtain the corresponding FAs, access to all guards and updating actions is needed. If we transform interacting EFAs separately to FAs, information is lost. *The transformation must consider all components simultaneously.*

In [40], it was shown how N EFAs with n variables can be transformed to N *location* FAs and n *variable* FAs, where:

$$\textbf{STS}(E_1\|\ldots\|E_N) = A_1\|\ldots\|A_{N+n}.$$

However, it has been observed that this transformation procedure can be very time consuming, especially, for models with many guards and actions [48]. In Paper 2, it is explained how EFAs are transformed to FAs, based on a similar approach to [40], but on the symbolic level using BDDs. The symbolic transformation will in most of the cases resolve the transformation issue in [48].

Basically, the transformation algorithm collects the information stored in the guards and actions, and builds two kinds of automata *variable automata* and *location automata*. The variable automata model the updating of the variables in *all* EFAs, and the location automata have the same structure as the original extended automata without considering the action functions. The composed model of all variable automata, denoted as $A^{\mathcal{V}}$, will model the updating of all variables simultaneously. We denote the location automaton of an EFA by A^{loc}.

Having $N = N_1+N_2$ EFAs, with N_1 sub-plants $E_{P_1},\ldots,E_{P_{N_1}}$ and a N_2 sub-specifications $E_{Sp_1},\ldots,E_{Sp_{N_2}}$, the corresponding plant FA A_P and specification FA A_{Sp} can be computed as follows:

$$\begin{aligned} A_P &= A^{loc}_{P_1}\|\ldots\|A^{loc}_{P_{N_1}}\|A^{\mathcal{V}},\\ A_{Sp} &= A^{loc}_{Sp_1}\|\ldots\|A^{loc}_{Sp_{N_2}}\|A^{\mathcal{V}}. \end{aligned}$$

Consequently, based A_P and A_{Sp}, the conventional SCT can be applied to the model. Recall that this procedure is performed symbolically using BDDs.

3.2 SCT of Timed DES

As stated earlier, we model TDES by TEFAs. In Section 3.1.1, we showed how SCT can be applied to EFAs by transforming them to their corresponding FAs.

In order to apply SCT to TEFAs, we transform TEFAs to EFAs by introducing an event $tick$ that will be be treated in a special manner.

3.2.1 Transformation of TEFAs to EFAs

As mentioned earlier, the evolvement of the clocks occur implicitly by the global digital clock. However, to addapt TEFAs to the conventional SCT, we need to have an explicit representation of the clocks. In particular, we need to somehow consider the global clock in the models. The global clock can be imagined as a function $tickcount : \mathbb{R}^+ \to \mathbb{N}$,

$$tickcount(t) = n, \ \ n \leq t < n + 1,$$

where $\mathbb{R}^+ = \{t \in \mathbb{R} | t \geq 0\}$ is the set of positive real values. Consequently, the temporal resolution available for modeling purposes is thus just one unit of clock time. For a TEFA, this behavior, can be represented by an EFA (consisting of only regular variables) by introducing an additional event $tick$ as in [45]. The event $tick$ occurs exactly at the real time moments, which can be imagined to be generated by the global clock. In Paper 3, it is shown how a TEFA can be transformed to its corresponding EFA, referred to as the $tick$-EFA. In the following, we briefly describe the transformation procedure.

Initially, the event $tick$ is added to the alphabet of the TEFA. For each clock c in the model with maximum value μmax, the clock is considered as a regular variable with domain $\{0, \ldots, \mu\text{max}\}$. For each invariant-free location l in the TEFA, the following transitions are added:

$$(l, tick, c < \mu\text{max}, c := c + 1, l) \ \text{ and}$$
$$(l, tick, c \geq \mu\text{max}, c := c, l).$$

This transition extension is performed for all clocks in the TEFA.

In the existence of an invariant for l, it should not be possible to execute the $tick$ event if the invariant is not satisfied. For instance, if the location l has an invariant $c \leq 3$, only a transition $(l, tick, c < 3, c := c + 1, l)$ should be added. Note that in the new $tick$ transition, $c \leq 3$ has been changed to $c < 3$; because based on the invariant semantics, c should not evolve when value 3 is reached. In general, a location l with invariant $Inv(l)$ can be described by the following $tick$ transition,

$$(l, tick, \widehat{Inv}(l), c := c + 1, l),$$

where $\widehat{Inv}(l)$ is obtained by replacing all terms in form of $c < \omega$, $c \leq \omega$, and $c == \omega$ appearing in $Inv(l)$ with $c < \omega - 1$, $c \leq \omega - 1$ and $c == \omega - 1$, respectively.

In the next section, we describe how the $tick$ event is treated from an SCT point of view.

3.2.2 Controllability of TDES

We base the theory of controllability for the $tick$-based models on the framework in [49], where the event $tick$ is treated in a special manner.

A new category of events that arises naturally in the presence of timing is the *forcible events*, $\Sigma^f \subseteq \Sigma \backslash \{tick\}$. A forcible event is one that can preempt a tick of the global clock. If at a given state of the plant, a $tick$ and one or more forcible events are enabled, then the SCT permits the effective erasure of $tick$ from the current list of enabled events. Notice that a forcible event may be controllable or uncontrollable; a forcible event that is uncontrollable cannot be directly prevented from occurring by disablement. By the given description of forcible events, the status of $tick$ lies intuitively between 'controllable' and 'uncontrollable': no technology could 'prohibit' tick in the sense of 'stopping the clock', although a forcible event, if it is enabled, may preempt it. However, to simplify terminology, in [49], $tick$ is considered to be controllable.

To define controllability for the $tick$ models, the definition of controllability of untimed DES (Definition 3.1) is extended. Let G and K be two DFAs. A state $(p, q) \in Q_G \times Q_K$ is controllable if,

- $\left(\Gamma_{G\|K}((p,q)) \cap \Sigma^f\right) \neq \emptyset$, then

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

- $\left(\Gamma_{G\|K}((p,q)) \cap \Sigma^f\right) = \emptyset$, then

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

Thus, K controllable means that an event σ (in the full alphabet Σ including $tick$) may occur in $G\|K$ if σ is currently enabled in G and either (i) σ is uncontrollable, or (ii) $\sigma = tick$ and no forcible event is currently enabled in $G\|K$. The effect of the definition is to allow the occurrence of $tick$ (when it is enabled in G) to be ruled out of $G\|K$ only when a forcible event is enabled in $G\|K$ and could thus (perhaps among other events in $\Sigma \backslash \{tick\}$) be relied on to preempt it. Notice, however, that a forcible event need not preempt the occurrence of competing non-tick events that are enabled simultaneously. In general the model will leave the choice of tick-preemptive transition nondeterministic. In the sequel, we refer to the states that become uncontrollable due to the elimination of $tick$, as *timed uncontrollable states*.

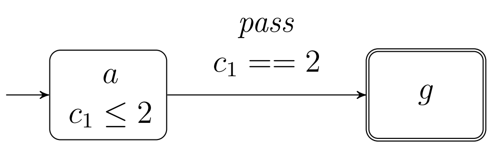
Notice that the introduction of the event $tick$ will not impact the 'nonblocking' definition for untimed DES (Definition 3.2).

In the following, we show an example taken from [49].

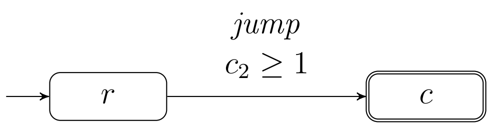
EXAMPLE 3.1 *Endangered Pedestrian*

Consider two TEFAs, shown in Figure 3.2a and 3.2b, representing a bus and a pedestrian. The TEFA **BUS** has a clock c_1 with domain $\{0, 1, 2, 3\}$ and **PED** has a clock c_2 with domain $\{0, 1, 2\}$. The bus can make a single transition $pass$ between the activities 'approaching' and 'gone by', and the pedestrian may make a single transition $jump$ from 'road' to 'curb'. We assume that the events $jump$ and $pass$ are controllable and uncontrollable, respectively. In addition, we assume that $jump$ is a forcible event. Suppose it is required that the pedestrian be saved, such that she jumps before the bus passes. The specification automaton of this requirement is shown in Figure 3.2c.

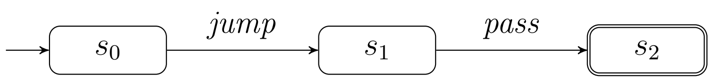
To apply the SCT of timed DES to this example, we first transform the TEFAs to their corresponding $tick$-EFAs, shown in Figure 3.3. Next, we transform the EFAs to DFAs.



(a) The TEFA **BUS**.



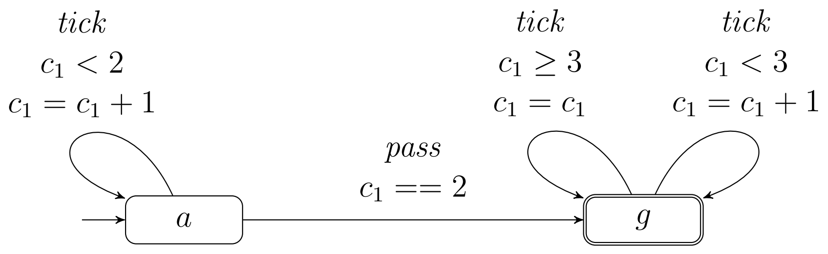
(b) The TEFA **PED**.



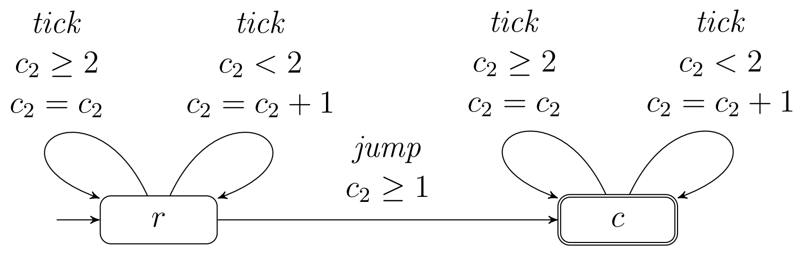
(c) The specification **SPEC**.

Figure 3.2: The TEFAs representing the plant and specification of Example 3.1.

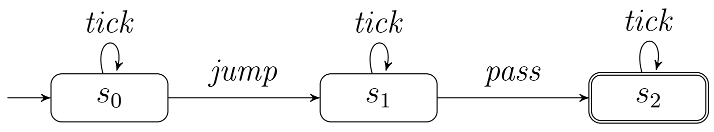
Figure 3.4 shows the corresponding DFA of **BUS**||**PED**||**SPEC**. In the DFA, a state is represented as $\langle(l_{\textbf{BUS}}, l_{\textbf{PED}}), (\mu_1^{\mathcal{C}}, \mu_2^{\mathcal{C}})\rangle$. For brevity, we have not included the location names of the specification in the figure. It can be observed that state $\langle(a, r), (2, 2)\rangle$ is uncontrollable because at this state the uncontrollable event $pass$ is enabled in the plant (the transition of **BUS**) but not in **BUS**||**PED**||**SPEC**. By removing this state, the supervisor is obtained. Notice that removing this state will disable the $tick$ event at $\langle(a, r), (1, 1)\rangle$, however, since the event $jump$ is forcible it can preempt the $tick$. □



(a) The $tick$-EFA of **BUS**.



(b) The $tick$-EFA of **PED**.



(c) The $tick$-EFA of **SPEC**.

Figure 3.3: The corresponding $tick$-EFAs of the TEFAs in Figure 3.2.

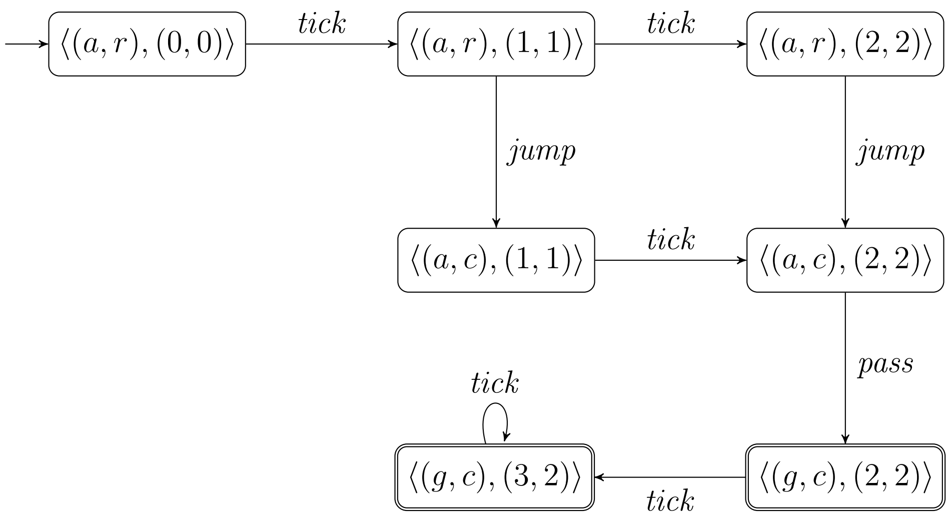

$\langle(a,r),(0,0)\rangle$
$tick$
$\langle(a,r),(1,1)\rangle$
$tick$
$\langle(a,r),(2,2)\rangle$
$jump$
$jump$
$\langle(a,c),(1,1)\rangle$
$tick$
$\langle(a,c),(2,2)\rangle$
$pass$
$tick$
$\langle(g,c),(3,2)\rangle$
$tick$
$\langle(g,c),(2,2)\rangle$

Figure 3.4: The corresponding DFA of **BUS**||**PED**||**SPEC**.

3.3 Synthesis

As stated earlier, the process of automatically computing the supervisor is called *synthesis*. Generally, the synthesis can be performed in two ways: *monolithic* or *structural*. In monolithic synthesis, a first candidate of the supervisor is obtained by computing the composed automaton $P \parallel Sp$, which we refer to as S_0 in the sequel. After the synthesis procedure, the forbidden states are removed from S_0, yielding the safe states [8, 50]. Having the safe states, the automaton representing the supervisor can be constructed. It is also possible to exploit the structure of the sub-plants and sub-specifications by considering the modularity properties of the system or using abstraction techniques [51–55]. This can improve the synthesis task considerably, because such algorithms usually cope with a smaller number of states. In this work, we compute the supervisor based on the monolithic approach. However, we will later show how we can represent the supervisor modularly by employing the monolithic supervisor.

Typically, the synthesis procedure is performed by *fixed point* computations, that is, starting from a set of states, extend the set iteratively with new states until a fixed point is reached, where no new states can be found. In the following, we first describe the conventional fixed point computations performed on untimed models. In the next part, we show how the fixed point computations can be modified to conform to the SCT for TDES.

Algorithm 1: SAFESTATESYNTHESIS

```
Input: A set of forbidden states $Q^x$
Output: The safe states
1 $i \leftarrow 0$;
2 $Q^x_0 \leftarrow Q^x$;
3 repeat
4     $i \leftarrow i + 1$;
5     $Q' \leftarrow \text{RESTRICTEDBACKWARD}(Q^m, Q^x_{i-1})$;
6     $Q'' \leftarrow \text{UNCONTROLLABLEBACKWARD}(Q \backslash Q')$;
7     $Q^x_i \leftarrow Q^x_{i-1} \cup Q''$ ;
  until $Q^x_i = Q^x_{i-1}$;
8 return $\text{RESTRICTEDFORWARD}(Q^x_i)$;
```

3.3.1 Untimed DES

Given an STS, modeled by FAs or EFAs, Algorithm 1 shows a simple algorithm for computing the safe states [27] for an untimed DES. The algorithm starts with a set of forbidden states Q^x, which is the union of the explicitly forbidden states and the *initially* uncontrollable states that can be computed based on Definition

3.1. Then, Q^x is iteratively extended by adding all states that can reach the forbidden states or the non-coreachable states in an uncontrollable manner until a fixed point is reached. To obtain a supervisor that only consist of reachable states, based on the extended set of forbidden states, a reachability computation is performed (Algorithm 4), finding all reachable states that do not contain any forbidden state. Note that based on SCT, a supervisor that contains unreachable states can also be considered as a correct supervisor, however, we remove the unreachable states for the purpose of this work, described later. The set Q is the universal set, that is, the cross product of all automata.

Algorithm 2 computes the set of coreachable states by avoiding any forbidden states given as input.

Algorithm 3 computes the set of states that can reach a set of forbidden states, given as input, by only executing uncontrollable events, yielding the uncontrollable states. In particular, if a state is forbidden in S_0, then all ingoing transitions to this state should be removed. Hence, if one of the ingoing transitions includes an uncontrollable event, it will be removed while the plant can execute it, which is the definition of an uncontrollable state.

Given a set of states $W \subseteq Q$, the set-based operator $\texttt{Image}(W, \mapsto)$ computes the set of states that can be reached by executing one transition, formally defined as:

$$\texttt{Image}(W, \mapsto) \triangleq \{\acute{q} \in Q | \exists q \in W : (q, \sigma, \acute{q}) \in \mapsto\}. \tag{3.2}$$

The operator $\texttt{PreImage}(W, \mapsto)$ computes the set of states that, by one transition, can reach a state in W, formally defined as below:

$$\texttt{PreImage}(W, \mapsto) \triangleq \{q \in Q | \exists \acute{q} \in W : (q, \sigma, \acute{q}) \in \mapsto\}. \tag{3.3}$$

The transition relation $\mapsto_{S_0}$ represents the entire transition relation of S_0, while $\overset{u}{\mapsto}_{S_0}$ includes only those transitions that consider the uncontrollable events.

Algorithm 2: RESTRICTEDBACKWARD

Input: A set of marked states Q^m, and a set of forbidden states Q^x
Output: The coreachable states

```
1 $i \leftarrow 0$;
2 $Q_0 \leftarrow Q^m \backslash Q^x$;
3 repeat
4     $i \leftarrow i + 1$;
5     $Q_i \leftarrow (Q_{i-1} \cup \texttt{PreImage}(Q_{i-1}, \mapsto_{S_0})) \backslash Q^x$;
  until $Q_i = Q_{i-1}$;
6 return $Q_i$;
```

Algorithm 3: UNCONTROLLABLEBACKWARD

```
    Input: A set of forbidden states $Q^x$
    Output: The uncontrollable states
1 $i \leftarrow 0$;
2 $Q_0^x \leftarrow Q^x$;
3 repeat
4     $i \leftarrow i+1$;
5     $Q_i^x \leftarrow Q_{i-1}^x \cup$ PreImage($Q_{i-1}^x, \overset{u}{\mapsto}_{S_0}$);
  until $Q_i^x = Q_{i-1}^x$;
6 return $Q_i^x$;
```

Algorithm 4: RESTRICTEDFORWARD

```
    Input: A set of initial states $Q^0$, and a set of forbidden states $Q^x$
    Output: The reachable states
1 $i \leftarrow 0$;
2 $Q_0 \leftarrow Q^0$;
3 repeat
4     $i \leftarrow i+1$;
5     $Q_i \leftarrow (Q_{i-1} \cup$ Image($Q_{i-1}, \mapsto_{S_0}$)$) \setminus Q^x$ ;
  until $Q_i = Q_{i-1}$;
6 return $Q_i$;
```

EXAMPLE 3.2

Consider a plant and a specification, shown in Figure 3.5, for which we will synthesize a supervisor. The alphabet of each automaton is its corresponding events shown in the figure. The only marked state in the system is s_3, which is illustrated by a double-line around the state. By convention, all states in the plant are supposed to be implicitly marked.

We apply Algorithm 1 to this example. As stated earlier, a fist candidate of the supervisor is the composed automaton $S_0 = \mathbf{P} \parallel \mathbf{SP}$, shown in Figure 3.5c. Initially, the system has one uncontrollable state (p_6, s_2), which will be the input to the algorithm, i.e., $Q^x = Q_0^x = \{(p_6, s_2)\}$. In this state the uncontrollable event u_2 is blocked by the supervisor, while it is enabled by the plant. In the first iteration, the sets Q', Q'', and Q_1^x are,

$$\begin{aligned}
Q' &= \textsc{RestrictedBackward}(\{(p_4, s_3)\}, \{(p_6, s_2)\}) = \\
&\qquad\qquad \{(p_4, s_3), (p_1, s_1), (p_0, s_0), (p_2, s_2)\}, \\
Q'' &= \textsc{UncontrollableBackward}(\{(p_6, s_2), (p_5, s_2), (p_3, s_1)\}) = \\
&\qquad\qquad \{(p_1, s_1), (p_6, s_2), (p_5, s_2), (p_3, s_1)\}, \\
Q_1^x &= Q_0^x \cup Q'' = \{(p_1, s_1), (p_6, s_2), (p_5, s_2), (p_3, s_1)\}
\end{aligned}$$

Since $Q_0^x \neq Q_1^x$, a fixed point has not been reached, and thus another iteration of SAFESTATESYNTHESIS will be carried out:

$$\begin{aligned}
Q' &= \textsc{RestrictedBackward}(\{(p_4, s_3)\}, Q_1^x) = \\
&\qquad\qquad \{(p_4, s_3), (p_0, s_0), (p_2, s_2)\}, \\
Q'' &= \textsc{UncontrollableBackward}(\{(p_1, s_1), (p_6, s_2), (p_5, s_2), (p_3, s_1)\}) = \\
&\qquad\qquad \{(p_1, s_1), (p_6, s_2), (p_5, s_2), (p_3, s_1)\}, \\
Q_x^2 &= Q_x^1 \cup Q'' = \{(p_1, s_1), (p_6, s_2), (p_5, s_2), (p_3, s_1)\}.
\end{aligned}$$

At this step, a fixed point is reached because $Q_1^x = Q_2^x$.

By performing RESTRICTEDFORWARD(Q_2^x) and removing Q_2^x from the reachable states in S_0, the safe states are computed, yielding:

$$Q^{safe} = \{(p_4, s_3), (p_0, s_0), (p_2, s_2)\}.$$

The supervisor is shown in Figure 3.5d. □

For a more formal and detailed explanation of the conventional supervisory synthesis, refer to [50, 56, 57].

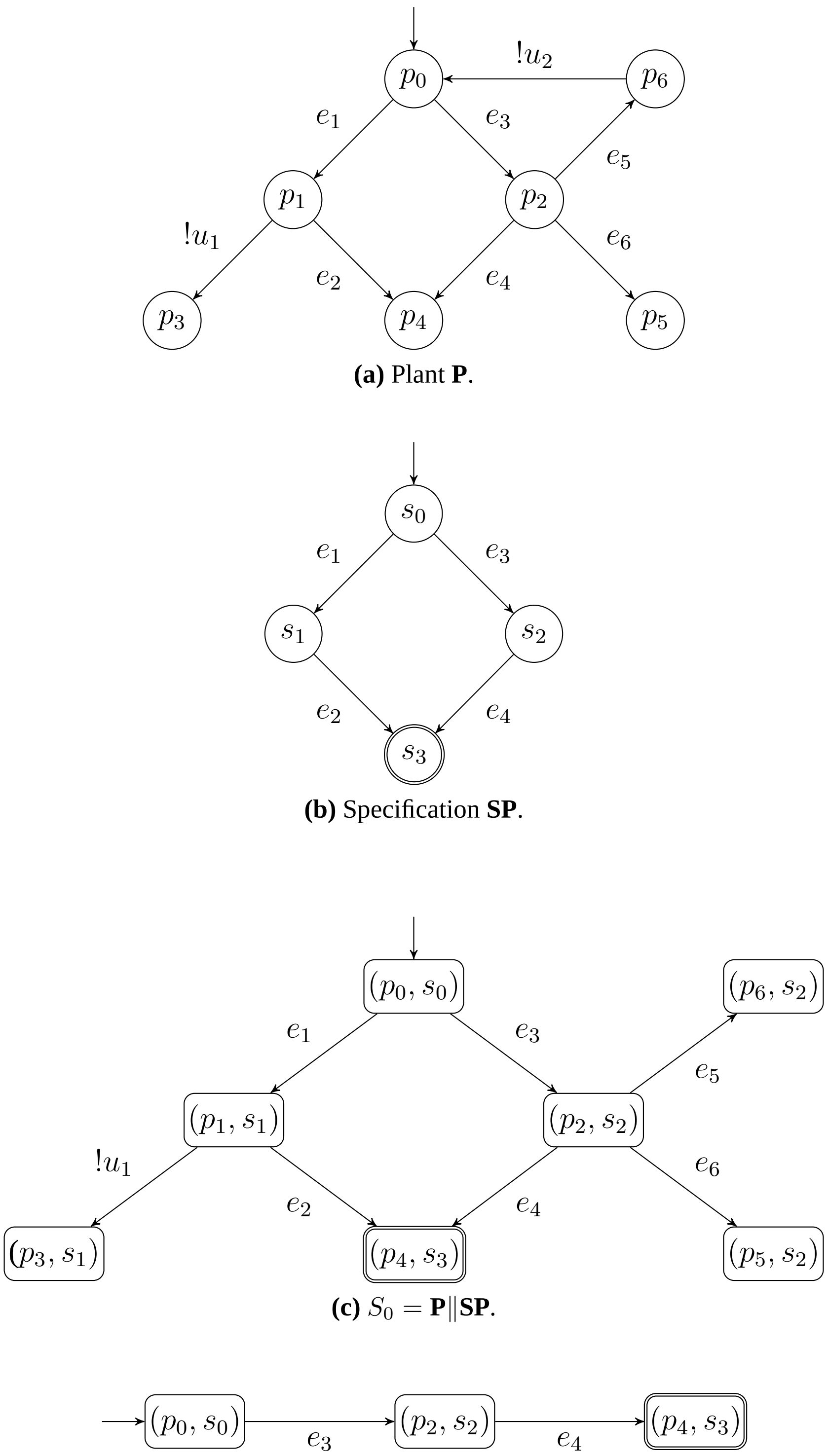
3.3.2 Timed DES

As pointed out in Section 3.2.2, the nonblocking analysis of TDES is exactly the same as for the untimed DES, described in the previous section. We will thus explain how the fixed point computation UNCONTROLLABLEBACKWARD (Algorithm 3) can be modified to conform with the definition of controllability of TDES. In particular, in addition to the uncontrollable states caused by uncontrollable events, we also need to find the timed uncontrollable states.

Algorithm 5 shows how the uncontrollable states computed in Algorithm 3 are extended with the timed uncontrollable states. The transition functions $\overset{tick}{\mapsto}_{S_0}$ and $\overset{f}{\mapsto}_{S_0}$, represent the transitions in S_0, which only include *tick* and forcible events, respectively. Given a set of states $W \subseteq Q$; $\texttt{Disabled}(W, \mapsto)$ computes the states that are not among the source-states of $\mapsto$, formally defined as below:

$$\texttt{Disabled}(W, \mapsto) \triangleq \{q \in W |\ \nexists(q, \sigma, \acute{q}) \in \mapsto\}. \tag{3.4}$$

In line 5, $\texttt{PreImage}(Q_{i-1}^x, \overset{tick}{\mapsto}_{S_0})$ computes the set of states that can reach a state in Q_{i-1}^x by executing a *tick* event. Among these states, those that do not have an outgoing forcible event are the timed uncontrollable states, $Q^{timedUnc}$. Notice that the initially uncontrollable states that will be passed to SAFESTATESYNTHESIS (Algorithm 1) should also include the initially timed uncontrollable states.



(a) Plant **P**.

(b) Specification **SP**.

(c) $S_0 = \mathbf{P} \| \mathbf{SP}$.

(d) The supervisor.

Figure 3.5: The plant, the specification, the composed model, and the supervisor for Example 3.2.

Algorithm 5: TICKUNCONTROLLABLEBACKWARD

```
   Input: A set of forbidden states $Q^x$
   Output: The uncontrollable states
1  $i \leftarrow 0$;
2  $Q^x_0 \leftarrow Q^x$;
3  repeat
4      $i \leftarrow i + 1$;
5      $Q^{timedUnc} \leftarrow$ Disabled(PreImage($Q^x_{i-1}, \overset{tick}{\mapsto}_{S_0}$), $\overset{f}{\mapsto}_{S_0}$);
6      $Q^x_i \leftarrow Q^x_{i-1} \cup$ PreImage($Q^x_{i-1}, \overset{u}{\mapsto}_{S_0}$) $\cup\, Q^{timedUnc}$;
   until $Q^x_i = Q^x_{i-1}$;
7  return $Q^x_i$;
```

Tick Elimination

The $tick$ models suffer from a major problem. The state size is very sensitive to the clock frequency: a $tick$ event must be associated with the passage of each unit of time. As the clock frequency increases, so must the number of $tick$ events. As a consequence, performing reachability analysis based on $tick$ models usually needs many iterations in the fixed point computations. In addition, as we will see in Chapter 4, in a BDD-based approach, the intermediate BDDs representing the reachable states can be very big, causing state space explosion. In the following, we explain how the iterations caused by the $tick$ event can be eliminated to tackle the aforementioned issues.

Consider a TDES modeled by TEFAs. The idea lies in the fact that time cannot be stopped. In tick-EFAs, this indicates that all the $tick$ transitions will eventually occur, unless there exists a location invariant. For instance, consider two clocks with domains $\{0, \ldots, 3\}$ and $\{0, \ldots, 5\}$ and assume $\langle \ell, 1, 2\rangle$ is the current state of the system. The sequence of the states that can be reached by the $tick$ event is:

$$\langle \ell, 1, 2\rangle \overset{tick}{\mapsto} \langle \ell, 2, 3\rangle \overset{tick}{\mapsto} \langle \ell, 3, 4\rangle \overset{tick}{\mapsto} \langle \ell, 3, 5\rangle.$$

Since all $tick$ transitions will eventually occur, it can be directly computed that when the state $\langle \ell, 1, 2\rangle$ is reached, the states $\{\langle \ell, 2, 3\rangle, \langle \ell, 3, 4\rangle, \langle \ell, 3, 5\rangle\}$ are also reachable. Given a set of states $W \subseteq Q$, we define the set-based operator $\texttt{TimedImage}(W)$ as below:

$$\texttt{TimedImage}(W) \triangleq \{\langle l, \mu^{\mathcal{V}}, \acute{\mu}^{\mathcal{C}}\rangle \mid \forall \langle l, \mu^{\mathcal{V}}, \mu^{\mathcal{C}}\rangle \in W : \\ \forall d \in D^{\mathcal{C}}_{\cup} : \acute{\mu}^{\mathcal{C}} = \varrho(\mu^{\mathcal{C}} + \mathbf{d})\}, \qquad (3.5)$$

where $\mu^{\mathcal{C}} + \mathbf{d} = (\mu^{\mathcal{C}}_1 + d, \ldots, \mu^{\mathcal{C}}_p + d)$. Essentially, the $\texttt{TimedImage}$ operator represents the *time evolution*. Similarly, we define $\texttt{TimedPreImage}(W)$ as below:

$$\texttt{TimedPreImage}(W) \triangleq \{\langle l, \mu^{\mathcal{V}}, \acute{\mu}^{\mathcal{C}}\rangle \mid \forall\langle l, \mu^{\mathcal{V}}, \mu^{\mathcal{C}}\rangle \in W : \\ \forall d \in D_{\cup}^{\mathcal{C}} : \acute{\mu}^{\mathcal{C}} = \varrho(\mu^{\mathcal{C}} - \mathbf{d})\}. \quad (3.6)$$

For brevity and simplicity, we write $(q, \sigma, \acute{Q})$ to denote a number of explicit transitions $\{(q, \sigma, \acute{q}_1), \ldots, (q, \sigma, \acute{q}_m)\}$, where $\acute{Q} = \{\acute{q}_1, \ldots, \acute{q}_m\}$. Based on the `TimedImage` operator, we propose the following definition.

Definition 3.3 Reachability Transition Relation

For a TEFA with transition relation $\rightarrow$*, its corresponding* reachability transition relation*, denoted by* $\rightarrowtail$*, is defined as below,*

$$\frac{(l, \sigma, g, \mathbf{a}, \acute{l}) \in\rightarrow \ \wedge \ (\mu^{\mathcal{V}}, \mu^{\mathcal{C}}) \models g \ \wedge \ \mu^{\mathcal{C}} \models Inv(l)}{(\langle l, \mu^{\mathcal{V}}, \mu^{\mathcal{C}}\rangle, \sigma, \acute{Q}) \in\rightarrowtail}, \quad (3.7)$$

where

$$\acute{Q} = \{\acute{q} \mid \forall \acute{q} \in \texttt{TimedImage}(\{\langle \acute{l}, \mathbf{a}^{\mathcal{V}}(\mu^{\mathcal{V}}, \mu^{\mathcal{C}}), \mathbf{a}^{\mathcal{C}}(\mu^{\mathcal{C}})\rangle\}) : \acute{q} \models Inv(\acute{l})\}.$$

Consequently, by using $\rightarrowtail$ in a fixed point computation, (as the transition relation passed to the `Image` and `PreImage` operators), rather than transitions based on tick-EFAs:

1. a number of states can be reached with a single iteration, compared to the $tick$ transitions, where multiple iterations are required (multiple calls of `Image` and `PreImage` operators);

2. usually the corresponding BDD of a set of states becomes smaller than the intermediate BDDs resulted after executing a $tick$ transition.

The elimination of the $tick$ event will not impact the correctness of the fixed point computations related to the nonblocking property. However, for controllability, since TICKUNCONTROLLABLEBACKWARD is based on the $tick$ event, we need a new way to compute the timed uncontrollable states.

By looking at Figure 3.6, we explain how the timed uncontrollable states can be computed, based on the reachability transition relation. The figure shows a sample path of S_0, starting from state 0, executing some events s and reaching state 1, and by the occurrence of some $tick$ events, it will end up in state 7, which is assumed to be forbidden due to some reason, e.g., uncontrollability. Let us assume that the event σ^f is the only forcible event going out among the states 2-7. Based on TICKUNCONTROLLABLEBACKWARD, it can be deduced that the timed uncontrollable states for this example are states 5 and 6. Since

7 is forbidden, it should be removed, causing state 6 to be uncontrollable and removing state 6 will cause state 5 to be uncontrollable. Notice that removing state 5 will not make state 4 uncontrollable because it has an outgoing forcible event. Also observe that the outgoing transitions from states 2 and 3 will not impact the timed uncontrollability. The general procedure of finding the timed uncontrollable states can be described as follows. For a forbidden state, say q^x, find the closest state, say q^f, that can reach the forbidden state by executing a number of $tick$ events (in the figure, this state is 4). The timed uncontrollable states are then $\big(\texttt{TimedPreImage}(\{q^x\})\backslash\texttt{TimedPreImage}(\{q^f\})\big)\backslash\{q^x\}$. For this example, we have $\{1,\ldots,7\}\backslash\{1,\ldots,4\}\backslash\{7\} = \{5,6\}$. Observe that since the timed uncontrollable states should eventually be removed from S_0, we can include the forbidden state q^x in the set of timed uncontrollable states, yielding $\texttt{TimedPreImage}(\{q^x\})\backslash\texttt{TimedPreImage}(\{q^f\})$.

Based on the aforementioned reasoning, in Paper 4, it is shown how the timed uncontrollable states can be computed according to a new fixed point algorithm.

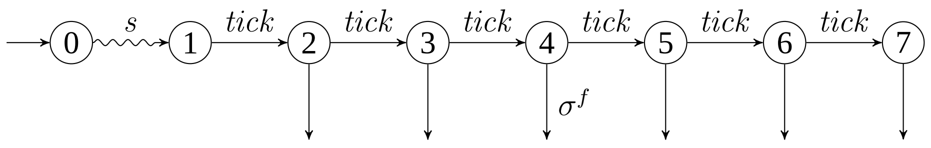


Figure 3.6: A sample path of S_0.

3.4 Supervisor Representation

So far, we have discussed how the supervisor is “computed” as a monolithic automaton. The next concern is how to “represent” the supervisor. This issue can be treated from two different perspectives: *modeling* and *implementation*.

Modeling

A typical issue that arises, when modeling a system modularly based on conventional SCT, is that for large and complex systems, representing the supervisor monolithically, may become untractable for the designers. More specifically, the designers retrieve the final supervisor as a black box, without clearly understanding why some events become disabled after the synthesis. Furthermore, after the synthesis, the designers will end up in a different scope, starting by a modular representation and ending in a monolithic one. This could be cumbersome if the designers later on desire to make some certain modifications in the specification.

Implementation

From another point of view, implementing a huge monolithic supervisor in a hardware may require more memory than available. Typically, a modular supervisor consumes less memory in a controller. The reason is that the synchronization will be performed online in the controller, see [57–59], which will alleviate the problem of exponential growth of the number of states in the synchronization. In addition, in industry, the controller is typically implemented based on other representations such as sequential final charts (SFCs), ladder diagrams, Gantt charts, and PERT charts, where the controller is mainly represented as logical constraints. Hence, to implement a monolithic supervisor in a controller, one should transform some parts of the automaton to logical constraints, which may not be straightforward.

To tackle the aforementioned issues, in this section, we discuss how the monolithic supervisor can be represented modularly by extracting guards from the safe states and restricting the plant by adding the guards to the original models. In this way,

1. the designers will remain in the modular scope, which makes it possible to easily perform modifications on the resulting supervisor, e.g., changing the specification,

2. it becomes possible to implement the supervisor in a modular manner, which could especially be beneficial for hierarchial approaches,

3. the final representation will be closer to the one typically used in the industry for implementing a controller.

The guards are generated based on the computed supervisor, discussed in the following.

3.4.1 Representing the Supervisor as Guards

Recall that the supervisor influences the plant by preventing it to execute some events in its current state, in order to avoid violations on the given specification. Accordingly, at any state in S_0, an event is either *allowed* or *forbidden* to occur, in order to end up in a state of the supervisor. It is also possible that the execution of an event at a state does not affect the synthesis result, e.g., if the state is not reachable. For each event σ, we can thus generate a guard based on the states of the DFA representing the supervisor, indicating when σ is allowed to be executed. Our goal is to make the generated guards as compact and comprehensible as possible for the designers.

Concerning the states that are retained or removed after the synthesis procedure, for each event σ, three *basic state sets* can be considered that form the basis for generating the guard:

1. the states, where σ must be enabled in order to end up in states that belong to the supervisor,
2. the states, where σ must be disabled in order to avoid ending up in states that were removed after the synthesis procedure,
3. the states, where enabling or disabling σ does not make any changes in the final supervisor.

In the sequel, each state set will be described formally and in more detail. In the following definitions, we use S to denote the DFA representing the supervisor.

Definition 3.4 Forbidden state set, $Q^{\sigma}_{\mathbf{f}}$

Forbidden state set, $Q^{\sigma}_{\mathbf{f}}$, *is the set of states in the supervisor where the execution of* σ *is defined for* S_0, *but not for the supervisor:*

$$Q^{\sigma}_{\mathbf{f}} = \{q \in Q^{safe} \mid \sigma \in \Gamma_{S_0}(q) \ \wedge \ \sigma \notin \Gamma_S(q)\}.$$

Definition 3.5 Allowed state set, $Q^{\sigma}_{\mathbf{a}}$

Allowed state set, $Q^{\sigma}_{\mathbf{a}}$, *is the set of states in the supervisor where the execution of* σ *is defined for the supervisor:*

$$Q^{\sigma}_{\mathbf{a}} = \{q \in Q^{safe} \mid \sigma \in \Gamma_S(q)\}.$$

Notice that, if $Q^{\sigma}_{\mathbf{a}}$ is restricted to a smaller set, the guard generated from this state set will disable σ on transitions where the target-state has been retained after the synthesis procedure; characterizing a supervisor which is not minimally restrictive. On the other side, if $Q^{\sigma}_{\mathbf{a}}$ is extended to a larger set, the generated guard will let σ to be executed on transitions, where the target-state has been removed after the synthesis procedure; characterizing a supervisor, which might be blocking or uncontrollable. In other words, for each event $\sigma \in \Sigma$, $Q^{\sigma}_{\mathbf{a}}$ 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 (an analogous argument can be given for $Q^{\sigma}_{\mathbf{f}}$). A similar explanation can be given for $Q^{\sigma}_{\mathbf{f}}$.

In order to obtain compact and simplified guards, inspired from the Boolean minimization techniques, we determine a set of states where executing σ will not impact the result of the synthesis and utilize these states to minimize the guards, referred to as the *don't care* states. The formal definition of don't care states is given below. In the following, for a state set Q_α, the complement of Q_α is denoted as $C(Q_\alpha) = Q \backslash Q_\alpha$.

Definition 3.6 Don't-care state set, $Q^{\sigma}_{\mathbf{dc}}$

Don't-care state set, $Q^{\sigma}_{\mathbf{dc}}$, *is the set of states where event* σ *could either be enabled or disabled, without having any impact on the supervisor. It is formally defined as* $Q^{\sigma}_{\mathbf{dc}} = C(Q^{\sigma}_{\mathbf{a}} \cup Q^{\sigma}_{\mathbf{f}})$.

From Definition 1.2 and Definition 1.1 it can be concluded that for a given event σ, the states that can impact the supervisor are only the states where σ *must* be allowed, $Q^\sigma_{\mathbf{a}}$, or forbidden, $Q^\sigma_{\mathbf{f}}$, to occur and the remaining states can be considered as don't-care. It can also be shown that $Q^\sigma_{\mathbf{dc}} = C(Q^\sigma) \cup C(Q^{safe})$; the proof is included in [60].

Guard generation

Recall that a system can be modularly modeled as a number of sub-plants and sub-specifications, which together form N automata $A_1, \dots, A_N$. Hence, a state $q_{S_0} \in Q_{S_0}$, is an N-tuple $(q_{A_1}, \dots, q_{A_N})$. For an event σ, the guard $G^\sigma : Q_{A_1} \times Q_{A_2} \times \dots \times Q_{A_N} \to \mathbb{B}$ is desired:

$$G^\sigma(q_{A_1}, \dots, q_{A_N}) = \begin{cases} \top & (q_{A_1}, \dots, q_{A_N}) \in Q^\sigma_{\mathbf{a}} \\ \bot & (q_{A_1}, \dots, q_{A_N}) \in Q^\sigma_{\mathbf{f}} \\ \text{don't care} & \text{otherwise} \end{cases}$$

where $\mathbb{B}$ is the set of Boolean values. In particular, σ is allowed to be executed from the state $(q_{A_1}, \dots, q_{A_N})$ if the guard is evaluated to $\top$.

Before showing how the guard is generated, we first show how a propositional formula representing a set of states can be computed. Let us assume that a sub-state q_{A_i} belonging to a specific automaton A_i can be extracted from q_{S_0} by the function $\Phi : (Q_{A_1} \times Q_{A_2} \times \dots \times Q_{A_N}) \times A_i \to Q_{A_i}$. Let $Q_\alpha \subseteq Q_{S_0}$. The following procedure shows how a propositional formula, representing Q_α, can be computed:

1. Introduce N new variables $\{q_{A_1}, q_{A_2}, \dots, q_{A_N}\}$ where $D^{\mathcal{V}}_i = Q_{A_i}$.

2. The corresponding propositional formula of Q_α, PF(Q_α), will be:

$$\text{PF}(Q_\alpha) : \bigvee_{q \in Q_\alpha} \left(\bigwedge_{i=1}^{N} \Big(q_{A_i} == \Phi(q, A_i) \Big) \right) \tag{3.8}$$

   where $==$ is the equality operator.

For the sake of brevity, having $q^k_{A_i}$ as a state belonging to A_i, we denote $\neg(q_{A_i} = q^k_{A_i})$ as $(q_{A_i} \neq q^k_{A_i})$.

Definition 3.7 Size of a propositional formula

The number of equality terms, which has either the form $(q_{A_i} = q^k_{A_i})$ or $(q_{A_i} \neq q^k_{A_i})$, in the propositional formula is referred to as the size *of the formula. We denote the size of a propositional formula p by $|p|$.*

The guards can now be generated either based on $Q^\sigma_{\mathbf{a}}$ denoted as $G^\sigma_{\mathbf{a}}$, or based on $Q^\sigma_{\mathbf{f}}$ denoted as $G^\sigma_{\mathbf{f}}$, by computing the corresponding propositional formulae, i.e., $G^\sigma_{\mathbf{a}} = \text{PF}(Q^\sigma_{\mathbf{a}})$ and $G^\sigma_{\mathbf{f}} = \neg\text{PF}(Q^\sigma_{\mathbf{f}})$.

Guard Simplification

From a modeling perspective, a *smaller* formula would typically be more readable and comprehensible. Furthermore, in many cases, the generated guards can be very big and memory-intensive, which could make it difficult to implement them in a hardware with limited amount of memory, such as microcontrollers. Our goal is to find the smallest guard. Inspired by minimization methods of Boolean functions, simplified guards can be obtained by utilizing the don't-care states and applying some heuristic techniques. This minimization is performed on the symbolic level, explained in Chapter 4. Since the minimization and specifically the guard generation, are carried out on a symbolic level, some information related to the structure of the automata may be lost. Sometimes, by utilizing the structure of the system, the guards can be simplified. Here, we briefly describe two heuristics that can be applied in an attempt to obtain smaller guards:

1. *Complement states (CS)*: Consider an automaton consisting of states Q and let $Q_\alpha \subseteq Q$. By considering the fact that the corresponding propositional formula of Q_α can be represented in two ways; either directly based on Q_α or based on its complement $C(Q_\alpha) = Q \backslash Q_\alpha$, we can make the conclusion that
$$|C(Q_\alpha)| < |Q_\alpha| \Rightarrow |\neg\text{PF}(C(Q_\alpha))| < |\text{PF}(Q_\alpha)|.$$
Informally, if the complement of Q_α has less states than Q_α itself, then the propositional formula computed based on $C(Q_\alpha)$ is smaller than the one based on Q_α.

2. *Independent states (IS)*: Consider an example, where there exist 4 automata, and let assume that for event σ the following holds
$$\begin{aligned} Q^\sigma_{\mathbf{a}} &= \{(q^1_{A_1}, q^1_{A_2}, q^1_{A_3}, q^1_{A_4}), (q^1_{A_1}, q^3_{A_2}, q^1_{A_3}, q^1_{A_4})\}, \\ Q^\sigma_{\mathbf{f}} &= \{(q^1_{A_1}, q^2_{A_2}, q^1_{A_3}, q^2_{A_4})\}, \text{ and} \\ G^\sigma_{\mathbf{f}} &= q_{A_1} \neq q^1_{A_1} \vee q_{A_2} \neq q^2_{A_2} \vee q_{A_3} \neq q^1_{A_3} \vee q_{A_4} \neq q^2_{A_4}. \end{aligned}$$
An interesting feature about this example is that the sub-state $q^2_{A_2}$ is not included in $Q^\sigma_{\mathbf{a}}$. Thus, it suffices to merely include $q_{A_2} \neq q^2_{A_2}$ in the guard without concerning about the other terms. In other words, if $q_{A_2} = q^2_{A_2}$, no matter what the current states of the other automata are, event σ should be disabled. In such a case, state $q^2_{A_2}$ is called an *independent state*. It can be concluded that if a state $q \in Q^\sigma_{\mathbf{a}} \cup Q^\sigma_{\mathbf{f}}$ includes an independent state $q^k_{A_i}$, it suffices to merely include the term based on $q^k_{A_i}$ in the corresponding propositional formula.

For a more detailed information about the simplification procedure and the symbolic computations, refer to Paper 1.

Simplifying $G^{\sigma}_{\mathbf{a}}$ by utilizing the don't-care states and the heuristic techniques, yields a new guard, which we refer to the *allowed guard* and denote it by $\mathcal{G}^{\sigma}_{\mathbf{a}}$. Similarly, the *forbidden guard* $\mathcal{G}^{\sigma}_{\mathbf{f}}$ can be defined.

Depending on the internal structure of a model, either the allowed or the forbidden guard can be smaller. In the implementation both guards are computed and the smallest one, referred to as the *adaptive guard* and denoted by $\mathcal{G}^{\sigma}_{\star}$, is given to the designer.

Guard Attachment

To obtain a modular representation of the supervisor, the generated guards can be attached to the original models. Since the supervisor merely can restrict the plant's controllable events, the guards are generated for controllable events. Based on the following procedure, the supervisor can be represented as a number of EFAs or TEFAs:

1. for each event $\sigma \in \Sigma^c$ in the model, compute $\mathcal{G}^{\sigma}_{\star}$,

2. for each automaton A_i, add variable q_{A_i}, holding the current state of the automaton, to the model,

3. for each transition in automaton A_i, add an action function that updates q_{A_i} to its new value, and

4. for each event $\sigma \in \Sigma^c$, attach $\mathcal{G}^{\sigma}_{\star}$ to all transitions that include σ.

Note that if a transition in the original model contains a guard, then in the last step the computed guard $\mathcal{G}^{\sigma}_{\star}$ will be logically conjuncted with the existing guard.

We summarize this section by applying the above procedure to an illustrative example.

EXAMPLE 3.3

Consider a resource booking problem where two "dumb" robots need to book two resources in opposite order in order to carry out their tasks, shown in Figure 3.7. The resources can be considered as spatial zones that are going to be entered by the robots. To avoid collisions, the robots should not occupy the zones simultaneously. Hence, each robot can enter a zone if it is not occupied. These zones are shown by two shaded areas in the figure. The tasks of Robot 1 and Robot 2 are to reach Zone 2 and Zone 1, respectively. By assuming that the robots work independently, the system will obviously stuck in a deadlock after that robot 1 and robot 2 have occupied zones 1 and 2, respectively. In such situation, robot 1 cannot enter Zone 2 because it is occupied by robot 2 and vice versa. We model this example and compute the guards based on the monolithic supervisor.

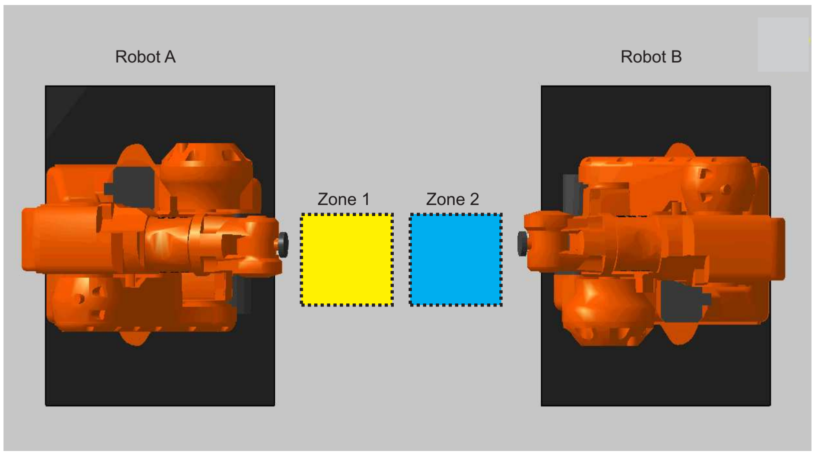
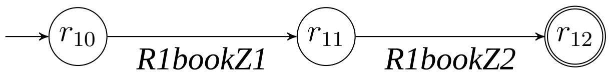
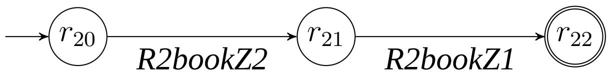


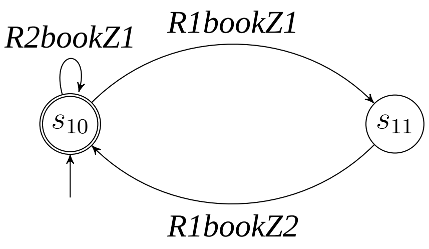
Figure 3.7: A robot cell consisting of two robots that book two resources in opposite order.



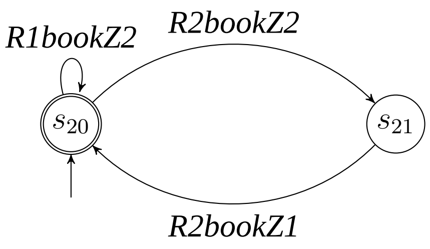
(a) Sub-plant **R1**.



(b) Sub-plant **R2**.



(c) Sub-specification **Z1**.



(d) Sub-specification **Z2**.

Figure 3.8: The automata modeling Example 3.3.

We model the robots' tasks as two sub-plants and the requirement of not colliding as two sub-specifications, shown in Figure 3.8. All the events are controllable. The reachable states of the composed automaton S_0 is shown in Figure 3.9. We can observe that the state $(r_{11}, r_{21}, s_{11}, s_{21})$ is blocking. By removing the blocking state from S_0, the supervisor is obtained.

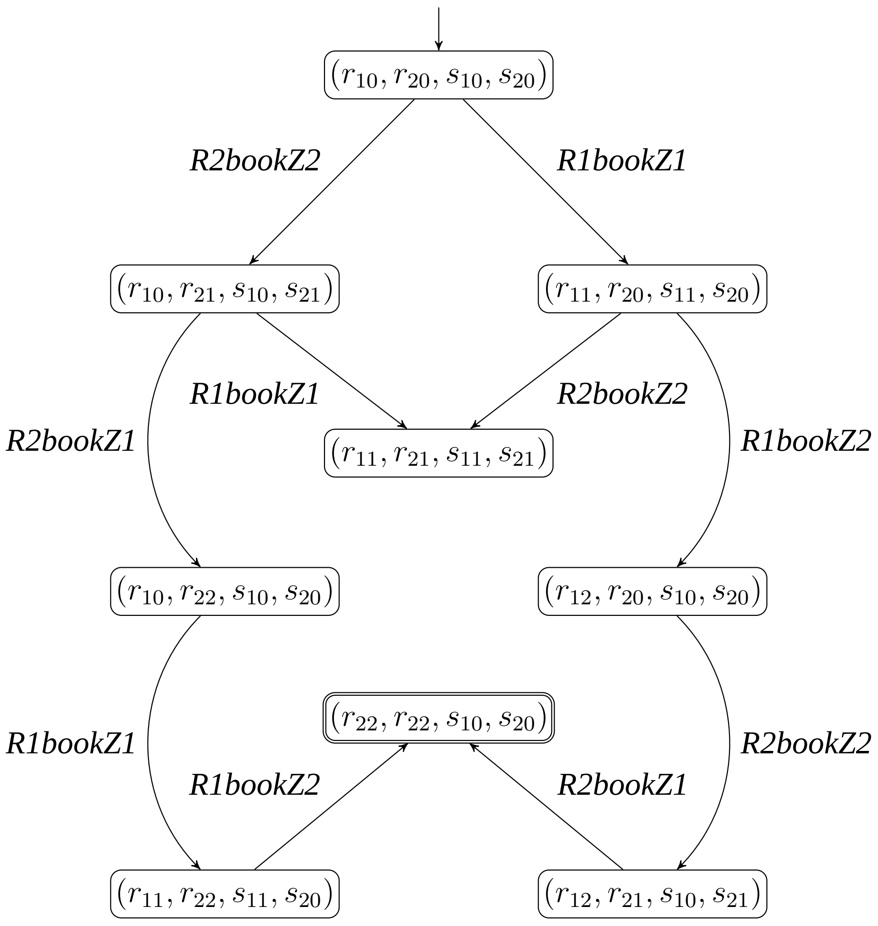

$(r_{10}, r_{20}, s_{10}, s_{20})$
R2bookZ2
R1bookZ1
$(r_{10}, r_{21}, s_{10}, s_{21})$
$(r_{11}, r_{20}, s_{11}, s_{20})$
R1bookZ1
R2bookZ2
R2bookZ1
R1bookZ2
$(r_{11}, r_{21}, s_{11}, s_{21})$
$(r_{10}, r_{22}, s_{10}, s_{20})$
$(r_{12}, r_{20}, s_{10}, s_{20})$
$(r_{22}, r_{22}, s_{10}, s_{20})$
R1bookZ1
R2bookZ2
R1bookZ2
R2bookZ1
$(r_{11}, r_{22}, s_{11}, s_{20})$
$(r_{12}, r_{21}, s_{10}, s_{21})$

Figure 3.9: The composed automaton S_0 for Example 3.3.

Let us compute $G_{\mathbf{f}}^{R1bookZ1}$. For the event $R1bookZ1$, the forbidden state set is $Q_{\mathbf{f}}^{R1bookZ1} = \{(r_{10}, r_{21}, s_{10}, s_{21})\}$. Hence,

$$G_{\mathbf{f}}^{R1bookZ1} = q_{\mathbf{R1}} \neq r_{10} \vee q_{\mathbf{R2}} \neq r_{21} \vee q_{\mathbf{Z1}} \neq s_{10} \vee q_{\mathbf{Z2}} \neq s_{21},$$

where the size is 4. Since $Q_{\mathbf{a}}^{R1bookZ1} = \{(r_{10}, r_{20}, s_{10}, s_{20}), (r_{10}, r_{22}, s_{10}, s_{20})\}$, we can conclude that s_{21} is an independent state. Thus, by applying the heuristic rule, we obtain $\mathcal{G}_{\mathbf{f}}^{R1bookZ1} = q_{\mathbf{Z2}} \neq s21$. This shows that event $R1bookZ1$ is not allowed to occur when the current state of automaton **Z2** is s_{21}, i.e., when Robot 2 has booked Zone 1. Note that an alternative guard could be $q_{\mathbf{R2}} \neq r_{21}$. Similarly, the guards for the other events can be computed. □

3.5 Related Work

Beside SCT, there exist other methods and theories for generating control functions for TDES. Among them, the one that is closely related to the SCT, is a game-theoretic approach based on *timed game automata (TGAs)* [61]. In UPPAAL [6, 62], the most well-known model checking tool, TGAs are used to model the systems. In this approach, the problem is modeled by two players, where player 1 (considered as the controller) executes controllable events, and player 2 (considered as the environment) executes uncontrollable events. The goal is to find a *strategy* (can considered as the supervisor in the SCT context), where player 1 should be guaranteed to reach a marked state, no matter what player 2 does. There are three main differences between the game-theoretic approach and the SCT. First, the synthesis theory for TGAs is based on states, while in SCT it is based on events. Second, in the SCT, it is guaranteed that a minimally restrictive supervisor is computed, while in the TGA-based approach the goal is find any strategy that ensures that a marked state is reached. Finally, in the SCT, the plant and specification are modeled by different types of automata, which will be the basis of the controllability definition, while the TGA-based approach define the controllability merely on the events, independent of what automata the uncontrollable events belong to. Hence, from a control point of view, the SCT defines controllability in a more natural manner.

Chapter 4

Symbolic Representation and Computation

As mentioned earlier, a system is typically modeled modularly by a number of sub-plants and sub-specifications. The global model is then obtained by composing the models. Having N automata $A_1, \ldots, A_N$, an upper bound for the number of states in the composed model is $\prod_{i=1}^{N} |Q^{A_i}|$, i.e., $|Q^{A_1 \| \ldots \| A_N}| \leq \prod_{i=1}^{N} |Q^{A_i}|$. By assuming that each automaton consists of k states, the upper bound will be k^N. This clearly indicates that the number of states of the composed model grows *exponentially* as the number of components increases. Therefore, the composed model for industrial applications with many components, could end up in a huge number of states, e.g., 10^{20} states. As a consequence, computing a supervisor for such systems could be a very time consuming and memory intensive process. In many cases, the number of states can exceed the amount of available hardware memory, which is known as the *state space explosion problem* and is the main complication when state-exploration methods are used for analysis of systems. This problem becomes more acute when the states are represented and enumerated *explicitly*, state by state.

Theoretically, the time complexity of synthesizing a nonblocking supervisor for a system is *NP-complete* [63, 64]. Hence, an approach that can compute a nonblocking supervisor in polynomial time is unlikely to be found. Nevertheless, various researchers have attacked this obstacle from different perspectives [52, 53, 65–67]. These approaches can be divided into two main categories. One way is to exploit the internal structure of the models such as modular and compositional synthesis [51, 54, 55]. However, most of them work under some preassumptions, which makes them unsuitable for our purposes such as guard generation and timing analysis. Another approach is to represent the states *symbolically (or implicitly)* by describing the state space and transitions by means of logical constraints. The main difference between explicit and symbolic representation is that in the former one the states are manipulated individually, while in the latter one *sets of states* are manipulated simultaneously. In addition, symbolic computations are typically carried out more efficiently compared to the explicit-state operations. It has been shown that symbolic techniques can allow significant gains in the size of systems that can be handled [27, 68].

In this thesis, all computations are "purely" carried out symbolically using *binary decision diagrams (BDDs)* [26], useful data structures for representing Boolean functions. It has been shown that BDD-based algorithms can improve the efficiency of synthesis dramatically [27, 30, 69]. For instance, in [27], the supervisor of a transfer line example with more than 10^{200} states was synthesized in few minutes.

4.1 Basics

Given a set of x Boolean variables $\mathcal{B}$, a Boolean function $f\colon \mathbb{B}^x \to \mathbb{B}$ ($\mathbb{B}$ is the set of Boolean values, i.e., 0 and 1) can be expressed using Shannon's decomposition [70]. This decomposition can be expressed by a directed acyclic graph, called *binary decision diagram (BDD)*, which consists of two types of nodes: *decision nodes* and *terminal nodes*. A terminal node can either be *0-terminal* or *1-terminal*, which corresponds to the resultant value of the function, i.e. 0 or 1. Each decision node is labeled by a Boolean variable and has two edges to its *low-child* and *high-child*, corresponding to assigning 0 and 1 to the variable, respectively. The *size* of a BDD, denoted as $|\mathbf{B}|$, refers to the number of decision nodes.

Using Shannon's decomposition [70], a BDD f can be recursively expressed as below

$$f = (\neg b \wedge f[0/b]) \vee (b \wedge f[1/b]) \qquad \text{for } \; b \in \mathcal{B},$$

where $f[0/b]$ and $f[1/b]$ refer to assigning 0 and 1 to all occurrences of the Boolean variable b, respectively. Furthermore, the notation $f[b'/b]$ is used to describe the result of substituting all free occurrences of b in f by b'.

A variable b_1 has a lower (higher) *order* than variable b_2 if b_1 is closer (or further) to the root and is denoted by $b_1 \prec b_2$ (or $b_2 \prec b_1$). If the variables in the BDD follow a total order, i.e. all variables occur in the same order on all paths, the BDD is called *Ordered BDD (OBDD)*. The variable ordering will impact the size of the BDD, however, finding an optimal variable ordering of a BDD is an NP-complete problem [71]. To find the optimal variable ordering is out of the scope of this thesis. In this work, all BDDs follow a fixed variable ordering, described later.

A BDD that fulfills the following conditions is referred to as *reduced BDD (RBDD)*:

1. no two distinct decision nodes have the same variable name and low- and high-children,

2. no decision node has identical low- and high-children.

The BDDs in this work are assumed to be both ordered and reduced, called *ROBDDs*. ROBDDs provide compact and canonical (unique) representation for a particular function and variable order [72]. Before reduction, the size of a BDD, is always exponential in the number of Boolean variables. This does not apply to ROBDDs, as they are sometimes reduced to "extremely compact" graphs.

Binary operations can be carried out efficiently on Boolean functions by applying tree operations on their corresponding ROBDDs. A binary operator $< \texttt{op} >$ between two BDDs f and g can be computed as

$$f < \texttt{op} > g = \Big(\neg b \wedge (f[0/b] < \texttt{op} > g[0/b])\Big) \vee \Big(b \wedge (f[1/b] < \texttt{op} > g[1/b])\Big).$$

If the operator is implemented based on dynamic programming, the time complexity of the algorithm will be $O(|f| \cdot |g|)$. Beside the compactness and efficiency of representing sets as BDDs, the set operations can also be simply implemented by BDDs. For instance, let BDDs f and g represent two state sets Q_1 and Q_2, respectively. Then, $Q_1 \cup Q_2$ and $Q_1 \backslash Q_2$ can be computed by $f \vee g$ and $f \wedge \neg g$, respectively. Note that the negation of a BDD is simply the substitution of 0-terminal and 1-terminal.

An operation that is used extensively in reachability analysis is the *existential quantification* operator over a Boolean variable b:

$$\exists b : f = f[0/b] \vee f[1/b].$$

The existential quantification can indeed be applied to a set of Boolean variables. Intuitively, the effect is that the variable b will be eliminated from the graph.

For a more elaborate and verbose exposition of BDDs and the implementation of different operators, refer to [73, 74].

To summarize, the power of BDDs lies in their simplicity and efficiency to perform binary operations, especially, when the BDDs have small sizes.

4.1.1 Characteristic Function

As stated earlier, in a symbolic representation, the computations are performed on sets of states. To this end, *characteristic functions* are used to to represent the corresponding BDDs of finite sets.

Definition 4.1 Characteristic function (CF)

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

$$\chi_Y(a) = \begin{cases} 1 & \textit{iff } a \in Y \\ 0 & \textit{otherwise} \end{cases} .$$

Since the set U is finite, in practice its elements are represented with numbers in $\mathbb{Z}_{|U|}$ or their corresponding binary x-tuples belonging to $\mathbb{B}^x$ ($x = \lceil \log_2^{|U|} \rceil$). For a binary CF, an injective function $\theta : U \to \mathbb{B}^x$ is used to map the elements in U to elements in $\mathbb{B}^x$. In general, $\chi_Y(a)$ is constructed as

$$\chi_Y(a) = \bigvee_{w \in Y} a \leftrightarrow \theta(w), \tag{4.1}$$

where $\leftrightarrow$ on two binary x-tuples $\mathbf{b_1}$ and $\mathbf{b_2}$ is defined as

$$\mathbf{b_1} \leftrightarrow \mathbf{b_2} \triangleq \bigwedge_{0 \le i < x} (b_{1i} \leftrightarrow b_{2i}), \tag{4.2}$$

where b_{ji} denotes the i-th element of b_j. In this way, different set-operations can be carried out on χ using basic Boolean operators.

In the sequel, all formal discussions will be based on the corresponding CFs of the BDDs. In the text, we will freely use "BDD" interchangeably with "characteristic function".

4.2 Representation of Models

In the following, we describe how DFAs and TEFAs can be symbolically represented by BDDs, i.e., how their corresponding CFs are computed.

4.2.1 Representation of DFAs

Reachability analysis on a DFA can be carried out based on its initial state and transition function. We define three tuples of Boolean variables $\mathbf{b}^Q$, $\acute{\mathbf{b}}^Q$, and $\mathbf{b}^\Sigma$, used to represent the source-states, target-states, and events of a transition, respectively. Note that, for the states, two Boolean tuples with different sets of Boolean variables are needed to distinguish between source-states and target-states. Hence, $|\mathbf{b}^Q| = |\acute{\mathbf{b}}^Q| = \lceil \log_2^{|Q|} \rceil$ and $|\mathbf{b}^\Sigma| = \lceil \log_2^{|\Sigma|} \rceil$. The automaton can then be represented as two BDDs for the initial state and the transition function

$$\begin{gathered} \chi_{\{q^0\}}(\mathbf{b}^Q) = \mathbf{b}^Q \leftrightarrow \theta(q^0) \\ \chi_{\mapsto}(\mathbf{b}^Q, \acute{\mathbf{b}}^Q, \mathbf{b}^\Sigma) = \bigvee_{(q,\sigma,\acute{q}) \in \mapsto} \chi_{(q,\sigma,\acute{q})}, \end{gathered} \tag{4.3}$$

where

$$\chi_{(q,\sigma,\acute{q})}(\mathbf{b}^Q, \acute{\mathbf{b}}^Q, \mathbf{b}^\Sigma) = \mathbf{b}^Q \leftrightarrow \theta(q) \;\wedge\; \acute{\mathbf{b}}^Q \leftrightarrow \theta(\acute{q}) \;\wedge\; \mathbf{b}^\Sigma \leftrightarrow \theta(\sigma). \tag{4.4}$$

In particular, first the BDD of each transition is created, and then all the BDDs are disjuncted to represent the total transition function.

Having N DFAs $A_1, \ldots, A_N$, the BDD representing the transition relation of $A = A_1 \| \ldots \| A_N$ can be computed in two steps. Since $\mathbf{b}^{\Sigma}$ is common in the CFs of all automata, first, we need to make all DFAs to have the same alphabet. To this end, for each DFA A_i and each $\sigma \in \Sigma_A \backslash \Sigma_{A_i}$, a self-loop transition is added to all states of A_i. The BDD of the synchronized model is then computed by conjuncting all BDDs representing the automata's transition relations, i.e., $\chi_{\mapsto_A} = \bigwedge_{i=1}^{N} \chi_{\mapsto_{A_i}}$.

For a DFA, we use a fixed variable ordering for its corresponding BDD that is based on the method presented in [75]. In this method, the variable ordering is influenced by the ordering of interacting automata, based on weighted search in their corresponding *process communication graph (PCG)*. 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 $|\Sigma^{A_1} \cap \Sigma^{A_2}|$. In some cases, the ordering can be improved [27].

4.2.2 Representation of TEFAs

Having a number TEFAs, in Section 3.3.2, we showed how the supervisor can be computed based on the corresponding reachability transition relation of the composed model, i.e., $\rightarrowtail_{S_0}$. In the following, the main idea for computing the corresponding BDD of $\rightarrowtail_{S_0}$ is given. For a detailed description of this procedure, refer to Paper 3.

Initially, the clocks of the TEFAs are treated as regular variables, yielding pure EFAs. Next, the BDD representing the composed model of the EFAs is computed. To consider the time semantics into the composed BDD, the target states $\acute{W}$ of all transitions are replaced by the states in $\texttt{TimedImage}(\acute{W})$, representing the time evolution. We denote the resulting BDD $\chi_{\rightarrowtail^{\cancel{\mathrm{Inv}}}_{S_0}}$. The BDD representing the reachability transition relation is obtained by conjucting $\chi_{\rightarrowtail^{\cancel{\mathrm{Inv}}}_{S_0}}$ with a BDD representing the invariants. The invariant BDD represents a set of pairs $\{(l, \mu^{\mathcal{C}}) \mid \mu^{\mathcal{C}} \models \mathit{Inv}(l)\}$.

In the following, we first describe how EFAs and their EFSC operator can be represented by BDDs; and second, we give the main idea how the BDD representing the time evolution can be computed. For a detailed description of this procedure, refer to Paper 4.

Representation of EFAs

The CF of the transition function of an EFA is represented based on its corresponding STS (Definition 2.5). Similar to the computation of DFAs, the CF is computed based on a set of Boolean variables $\mathbf{b}^{L}$, $\mathbf{b}_i^{\mathcal{V}}$, $\acute{\mathbf{b}}^{L}$, $\acute{\mathbf{b}}_i^{\mathcal{V}}$, and $\mathbf{b}^{\Sigma}$, used to represent the source-locations, current values of variable v_i, target-locations, updated values of variable v_i, and the events, respectively. The CF of a single transition $(l, \sigma, g, \mathbf{a}, \acute{l}) \in\to$ will thus be,

$$\begin{aligned}
&\chi_{(l,\sigma,g,\mathbf{a},\acute{l})}(\mathbf{b}^{\mathcal{V}}_1, \ldots, \mathbf{b}^{\mathcal{V}}_n, \acute{\mathbf{b}}^{\mathcal{V}}_1, \ldots, \acute{\mathbf{b}}^{\mathcal{V}}_n, \mathbf{b}^L, \acute{\mathbf{b}}^L, \mathbf{b}^\Sigma) = \\
&\qquad \Big(\bigvee_{\mu^{\mathcal{V}} \models g} \bigwedge_{i=1}^{n} \mathbf{b}^{\mathcal{V}}_i \leftrightarrow \theta(\mu^{\mathcal{V}}_i) \ \wedge \ \acute{\mathbf{b}}^{\mathcal{V}}_i \leftrightarrow \theta(a^{\mathcal{V}}_i(\mu^{\mathcal{V}}_i)) \Big) \wedge \\
&\qquad \mathbf{b}^L \leftrightarrow \theta(l) \ \wedge \ \acute{\mathbf{b}}^L \leftrightarrow \theta(\acute{l}) \ \wedge \ \mathbf{b}^\Sigma \leftrightarrow \theta(\sigma). \qquad (4.5)
\end{aligned}$$

In our framework, we assume that overflows on variables 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 explicit transition relation of an EFA E will be

$$\chi_{\mapsto_E} = \bigvee_{(l,\sigma,g,\mathbf{a},\acute{l}) \in \to} \chi_{(l,\sigma,g,\mathbf{a},\acute{l})} \ \wedge \bigwedge_{i=1}^{n} \chi_{D^{\mathcal{V}}_i}(b^{\mathcal{V}}_i) \wedge \bigwedge_{i=1}^{n} \chi_{D^{\mathcal{V}}_i}(\acute{b}^{\mathcal{V}}_i). \qquad (4.6)$$

The following example shows how the transition function of an EFA can be represented by a BDD.

EXAMPLE 4.1

Consider a nim game with 5 sticks on a table, and two players that take turn by removing one or two sticks. The winner is the player that takes the last stick(s). Fig. 4.1 depicts the EFA model for this game.

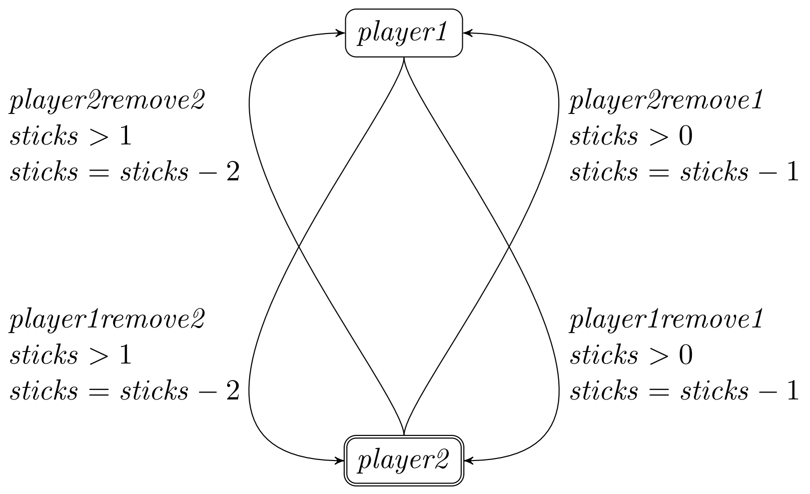


Figure 4.1: The EFA model for Example 2.1.

Fig. 4.2 shows the corresponding transition function for the EFA shown in Fig. 4.1. Note that the BDD does not contain the cases where *sticks* < 0 and *sticks* $>$

5. The BDD variables in the figure are labeled with numbers as follows

$$\begin{aligned}
\mathbf{b}^{\Sigma} &= (b_1^{\Sigma}, b_0^{\Sigma}) = (\text{‘1’}, \text{‘0’}),\\
\mathbf{b}^{L} &= (b_0^{L}) = (\text{‘2’}),\\
\acute{\mathbf{b}}^{L} &= (\acute{b}_0^{L}) = (\text{‘3’}),\\
\mathbf{b}^{sticks} &= (b_3^{sticks}, b_2^{sticks}, b_1^{sticks}, b_0^{sticks}) = (\text{‘7’}, \text{‘6’}, \text{‘5’}, \text{‘4’}),\\
\acute{\mathbf{b}}^{sticks} &= (\acute{b}_3^{sticks}, \acute{b}_2^{sticks}, \acute{b}_1^{sticks}, \acute{b}_0^{sticks}) = (\text{‘11’}, \text{‘10’}, \text{‘9’}, \text{‘8’}),
\end{aligned}$$

where b_0 is the least significant bit. Note that since the integers are represented in two's complement, four Boolean variables are used to represent $sticks$ because of the sign-bit. The location and event encoding is shown in Table 1.

Table 4.1: Event and location encoding for the EFA in Fig. 2.

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)		

For instance, let us track the transition

$$(\mathit{player2},\ \mathit{player2remove2},\ \mathit{sticks} > 1,\ \mathit{sticks} = \mathit{sticks} - 2,\ \mathit{player1})$$

on the BDD in Fig. 3. Event $\mathit{player2remove2}$ is identified by starting from node ‘0’, following the high-child to node ’1’ and following the high-child to node ‘2’, i.e. $b_1^{\Sigma} \wedge b_0^{\Sigma}$. The location $\mathit{player2}$ is identified by following the high-child from node ‘2’, i.e. b_0^L, and location $\mathit{player1}$ is identified by following the low-child from node ‘1’, i.e. $\neg \acute{b}_0^L$. The guard and action are identified by all the paths from node ‘3’ to node ‘11’.

As it can be observed, the BDD in this example is larger than its corresponding EFA, however, for larger models the BDDs typically become much more compact. □

We denote the CF, where the Boolean variables $\acute{\mathbf{b}}^{\mathcal{V}}$ have been removed by $\chi'_{(l,\sigma,g,\mathbf{a}\acute{l})}$:

$$\chi'_{(l,\sigma,g,\mathbf{a}\acute{l})}(\mathbf{b}_1^{\mathcal{V}}, \dots, \mathbf{b}_n^{\mathcal{V}}, \mathbf{b}^L, \acute{\mathbf{b}}^L, \mathbf{b}^{\Sigma}) = \exists \acute{\mathbf{b}}^{\mathcal{V}} : \chi_{(l,\sigma,g,\mathbf{a},\acute{l})}.$$

Having $N \geq 2$ EFAs $E_1, \dots, E_N$, similar to the transformation of EFAs to FAs, described in Section 3.1.1, the CF of the explicit transition function of $E = E_1 \| \dots \| E_N$, $\chi_{\mapsto_E}$, can be computed in three steps:

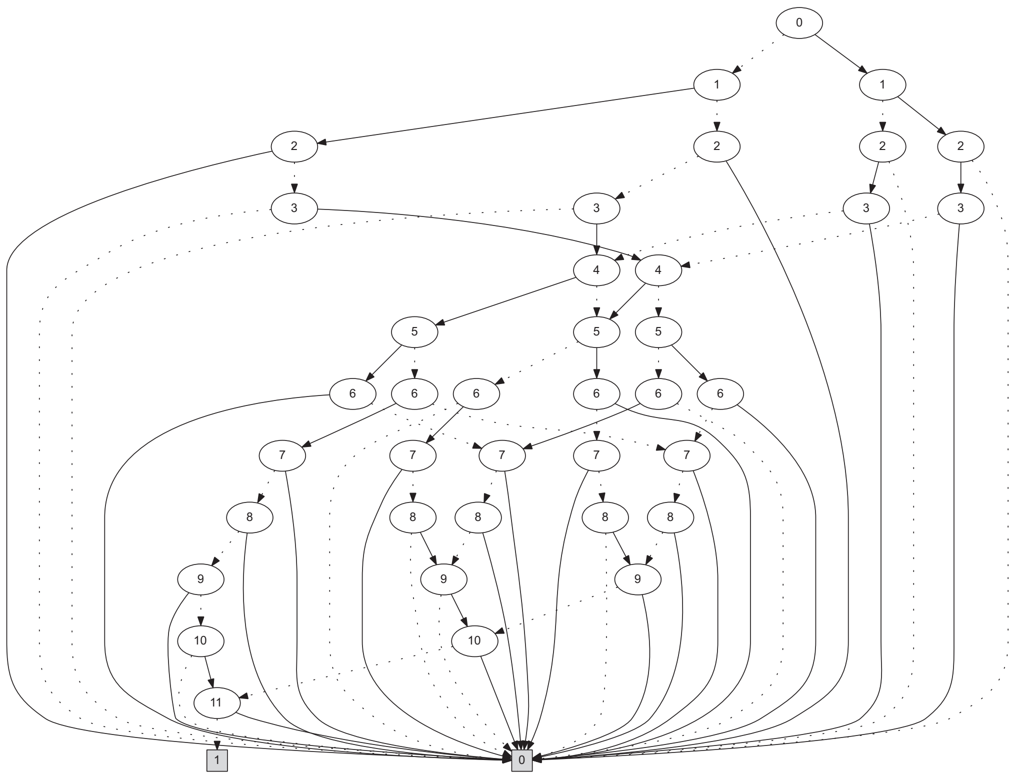


Figure 4.2: The corresponding BDD for the transition function of the EFA in Fig. 2.

1. Compute a CF, representing $\mapsto_E$ without including the actions, $\chi'_{\mapsto_E}$. This CF can be compared to the variable automaton A^{loc}, pointed out in Section 3.1.1.

2. Compute a CF, representing the update of the EFA variables, $\chi_{\mapsto^{\mathcal{V}}_E}$. This CF can be compared to the variable automaton $A^{\mathcal{V}}$, pointed out in Section 3.1.1.

3. Based on $\chi'_{\mapsto_E}$ and $\chi_{\mapsto^{\mathcal{V}}_E}$, compute $\chi_{\mapsto_E} = \chi'_{\mapsto_E} \wedge \chi_{\mapsto^{\mathcal{V}}_E}$.

As discussed earlier in (3.1), note that the result will be incorrect if steps 1 and 2 are carried out in a single step:

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

The procedure of computing the aforementioned CFs is presented in Paper 2.

Time Consideration

A stated earlier, having the BDD representing the composed model of the isomorphic EFAs, denoted as $\chi_{\mapsto_{S_0}}$, the time evolution is computed by replacing each target state by a set of states, representing the states that can be reached by the passage of time. We define the *timed transition relation* $\dashrightarrow$, where a tuple of clock evaluations $\mu^{\mathcal{C}}$ is expanded to the clock evaluations $\acute{\mu}^{\mathcal{C}}$ that can be reached by the passage of time:

$$\dashrightarrow = \{(\mu^{\mathcal{C}}, \acute{\mu}^{\mathcal{C}}) \mid \forall \mu^{\mathcal{C}} \in D^{\mathcal{C}} : \ \forall d \in D^{\mathcal{C}}_{\cup} : \ \acute{\mu}^{\mathcal{C}} = \varrho(\mu^{\mathcal{C}} + \mathbf{d})\}.$$

Introducing a set of temporary Boolean variables $\hat{\mathbf{b}}$, the corresponding BDD of the timed transition relation can be computed :

$$\chi_{\dashrightarrow}(\acute{\mathbf{b}}^{\mathcal{C}}_1, \ldots, \acute{\mathbf{b}}^{\mathcal{C}}_n, \hat{\mathbf{b}}^{\mathcal{C}}_1, \ldots, \hat{\mathbf{b}}^{\mathcal{C}}_n) =$$
$$\bigvee_{\mu^{\mathcal{C}} \in D^{\mathcal{C}}} \Big(\bigwedge_{i=1}^{p} \acute{\mathbf{b}}^{\mathcal{C}}_i \leftrightarrow \theta(\mu^{\mathcal{C}}_i) \ \wedge \bigvee_{d=0}^{|D^{\mathcal{C}}_{\cup}|} \bigwedge_{j=1}^{p} \hat{\mathbf{b}}^{\mathcal{C}}_j \leftrightarrow \theta(\varrho(\mu^{\mathcal{C}}_j + d)) \Big).$$

Based on $\chi_{\dashrightarrow}$, $\chi_{\rightarrowtail^{\cancel{\mathrm{Inv}}}_{S_0}}$ can be computed:

$$\chi_{\rightarrowtail^{\cancel{\mathrm{Inv}}}_{S_0}} = \Big(\exists \acute{\mathbf{b}}^{\mathcal{C}} : (\chi_{\mapsto_{S_0}} \wedge \chi_{\dashrightarrow})\Big)[\acute{\mathbf{b}}^{\mathcal{C}} / \hat{\mathbf{b}}^{\mathcal{C}}].$$

Essentially, in Paper 3, we show how the saturation function ϱ and the synchronization between the clocks, i.e., $\mu^{\mathcal{C}} + \mathbf{d}$, are implemented symbolically using BDDs.

4.3 Symbolic Synthesis

In the following, we describe how the conventional synthesis based on untimed DES (explained in Section 3.3.1) can be performed symbolically by BDDs. For symbolic synthesis of timed DES, refer to Paper 4.

In Section 3.3, we showed how the synthesis can be carried out based on fixed point computations. Basically, each algorithm starts by an initial state set and iteratively extends the set by the `Image` or `PreImage` operator until a fixed point is reached. Earlier, we showed how a transition function and a set of states can be represented by BDDs based on their corresponding CFs. The main issue that remains is the BDD implementation of the operators `Image` (2) and `PreImage` (3). Algorithm 6 shows the BDD-based implementation of the `Image` operator. The BDDs $\mathbf{B}_W$ and $\mathbf{B}_{\mapsto_{S_0}}$ represent a set of states W and the transition function of S_0, respectively. The BDD $(\mathbf{B}_{\mapsto_{S_0}} \wedge \mathbf{B}_W)$ represents all transitions, where their source-states are included in W. Consequently, $\exists \mathrm{b}^Q, \mathrm{b}^{\Sigma} : (\mathbf{B}_{\mapsto_{S_0}} \wedge \mathbf{B}_W)$ will represent all target-states that can be reached from states in W. Finally, in $\mathbf{B}_{nextStates}[\mathrm{b}^Q / \acute{\mathrm{b}}^Q]$, the Boolean variables representing the target-states will be substituted by their corresponding source-state variables.

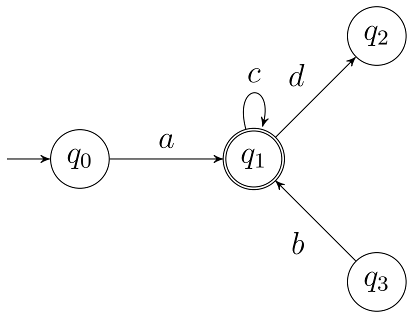


Figure 4.3: A sample automaton.

Algorithm 6: SYMBOLICIMAGE

```
Input: $\mathsf{B}_W$, $\mathsf{B}_{\mapsto_{S_0}}$
Output: The corresponding BDD for Image$(W, \mapsto_{S_0})$
1 $\mathsf{B}_{nextStates} \leftarrow \exists \mathbf{b}^Q, \mathbf{b}^\Sigma : (\mathsf{B}_{\mapsto_{S_0}} \wedge \mathsf{B}_W)$;
2 return $\mathsf{B}_{nextStates}[\mathbf{b}^Q/\acute{\mathbf{b}}^{\mathbf{Q}}]$;
```

For the `PreImage` operator, we first define the *backward transition relation* for $\mapsto$ as $\mapsfrom = \{(\acute{q}, \sigma, q) \mid (q, \sigma, \acute{q}) \in \mapsto\}$. The corresponding BDD of $\mapsfrom$, denoted by $\mathbf{B}_{\mapsfrom}$, can be computed by substituting the source and target variables in $\mathbf{B}_{\mapsto}$ with three BDD operations

$$\begin{aligned}
\mathbf{B}_1 &= \mathbf{B}_{\mapsto}[\grave{\mathbf{b}}^Q, \mathbf{b}^Q],\\
\mathbf{B}_2 &= \mathbf{B}_1[\mathbf{b}^Q, \acute{\mathbf{b}}^Q],\\
\mathbf{B}_{\mapsfrom} &= \mathbf{B}_2[\acute{\mathbf{b}}^Q, \grave{\mathbf{b}}^Q],
\end{aligned}$$

where $\grave{\mathbf{b}}^{\mathbf{Q}}$ is a new set of Boolean variables that is temporally used during the substitutions. The `PreImage` operator can then simply be implemented using Algorithm 6 by passing $\mathbf{B}_{\mapsfrom_{S_0}}$ to the routine rather than $\mathbf{B}_{\mapsto_{S_0}}$.

EXAMPLE 4.2

Let synthesize the automaton shown in Figure 4.3, representing S_0 for a sample system, by using BDD operations. It is assumed that all the events are controllable. Based on the state encoding in Table 4.2, we have:

$$\begin{aligned}
\chi_{\{q^0\}}(\mathbf{b}^Q) &= \neg b_1^Q \wedge \neg b_0^Q,\\
\chi_{Q^m}(\mathbf{b}^Q) &= \neg b_1^Q \wedge b_0^Q,
\end{aligned}$$

$$\begin{aligned}\chi_{\mapsto}(\mathbf{b}^Q, \acute{\mathbf{b}}^Q) &= (\neg b_1^Q \wedge \neg b_0^Q \wedge \neg \acute{b}_1^Q \wedge \acute{b}_0^Q) \vee (\neg b_1^Q \wedge b_0^Q \wedge \acute{b}_1^Q \wedge \neg \acute{b}_0^Q) \\ &\vee (b_1^Q \wedge b_0^Q \wedge \neg \acute{b}_1^Q \wedge \acute{b}_0^Q) \vee (\neg b_1^Q \wedge b_0^Q \wedge \neg \acute{b}_1^Q \wedge \acute{b}_0^Q), \\ \chi_{\leftarrow\!\shortmid}(\mathbf{b}^Q, \acute{\mathbf{b}}^Q) &= (\neg \acute{b}_1^Q \wedge \neg \acute{b}_0^Q \wedge \neg b_1^Q \wedge b_0^Q) \vee (\neg \acute{b}_1^Q \wedge \acute{b}_0^Q \wedge b_1^Q \wedge \neg b_0^Q) \\ &\vee (\acute{b}_1^Q \wedge \acute{b}_0^Q \wedge \neg b_1^Q \wedge b_0^Q) \vee (\neg \acute{b}_1^Q \wedge \acute{b}_0^Q \wedge \neg b_1^Q \wedge b_0^Q).\end{aligned}$$

Table 4.2: State encoding table for the automaton in Figure 4.3.

State	(b_1^Q, b_0^Q)
q_0	(0,0)
q_1	(0,1)
q_2	(1,0)
q_3	(1,1)

Table 4.3: Fixed point computation carried out by SAFESTATESYNTHESIS.

i	Q'	$\chi_{Q'}(\mathbf{b}^Q)$	Q_i^x	$\chi_{Q_i^x}(\mathbf{b}^Q)$
0	$\{\}$	$\perp$	$\{\}$	$\perp$
1	$\{q_0, q_1, q_3\}$	$\neg b_1^Q \vee b_0^Q$	$\{q_2\}$	$b_1^Q \wedge \neg b_0^Q$
2	$\{q_0, q_1, q_3\}$	$\neg b_1^Q \vee b_0^Q$	$\{q_2\}$	$b_1^Q \wedge \neg b_0^Q$

Table 4.4: Fixed point computation carried out by RESTRICTEDBACKWARD.

i	Q_i	$\chi_{Q_i}(\mathbf{b}^Q)$
0	$\{q_1\}$	$\neg b_1^Q \wedge b_0^Q$
1	$\{q_0, q_1, q_3\}$	$\neg b_1^Q \vee b_0^Q$
2	$\{q_0, q_1, q_3\}$	$\neg b_1^Q \vee b_0^Q$

We now perform SAFESTATESYNTHESIS(Q^x) (Algorithm 1), where Q^x is empty as there does not exists any explicitly forbidden state. Since the automaton does not contain uncontrollable events, UNCONTROLLABLEBACKWARD can be skipped from the algorithm and thus $Q'' = Q \backslash Q'$ in all iterations. Table 4.3 shows the elements and the characteristic function of Q' and Q_i^x for different iterations in the fixed point computations.

Table 4.4 shows the fixed point computation in RESTRICTEDBACKWARD, shown in Algorithm 2, that is carried out in the first and second iteration of SAFESTATESYNTHESIS.

Table 4.5: Fixed point computation carried out by RESTRICTEDFORWARD.

i	Q_i	$\chi_{Q_i}(\mathbf{b}^Q)$
0	$\{q_0\}$	$\neg b_1^Q \wedge \neg b_0^Q$
1	$\{q_0, q_1\}$	$\neg b_1^Q$
2	$\{q_0, q_1\}$	$\neg b_1^Q$

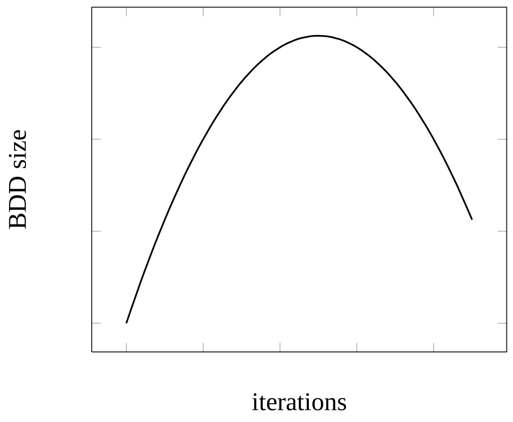


Figure 4.4: The typical pattern of the size of intermediate BDDs during the fixed point computations of reachability analysis.

Finally, by having $Q_2^x = \{q_2\}$, the safe states can be computed by calling RESTRICTEDFORWARD$(\{q_2\})$, shown in Algorithm 4. The fixed point computation is shown in Table 4.5. Consequently, the reachable safe states will be $\{q_0, q_1\}$. $\square$

4.3.1 Size of Intermediate BDDs

Typically, the size of the intermediate BDDs computed in each iteration of a fixed point computation for reachability analysis, follows a common pattern, shown in Figure 4.4. The important point that can be concluded from this figure is: the size of the BDD representing the fixed point is typically smaller than the maximum size that the intermediate BDDs can reach. Hence, even though there may exist enough memory to represent the final BDD, it is not sure that the intermediate BDDs can be computed. This is the main reason why eliminating the $tick$ event in the fixed point computations of TDES can be better. In particular, by reaching a number of states in one iteration (by the `TimedImage` operator, defined in (5)), the computation of the intermediate BDDs in the $tick$-based fixed point computations (obtained by executing the $tick$ event in each iteration) can be avoided.

4.4 Symbolic Guard Generation

In Section 3.4, we described how the guards, representing the supervisor, can be generated based on the basic state sets. The process of symbolic generation of a guard for an event can be divided into three consequent steps:

1. compute the corresponding BDDs for the basic state sets,
2. convert the BDDs to integer decision diagrams (IDDs),
3. generate the guard based on the IDDs.

We describe each step separately.

4.4.1 Symbolic Computation of the Basic State Sets

The first step of generating the guard is to compute the corresponding BDDs for the basic state sets, as described in Section 3.4. The corresponding BDD of S_0's transition function is used as the basis for generating these state sets. For an event σ, we first compute the BDD representing the states from which σ is enabled, denoted by Q^{σ}:

$$\chi_{Q^{\sigma}} = \exists \acute{\mathbf{b}}^{Q}, \mathbf{b}^{\Sigma} : \ \chi_{\mapsto_{S_0}} \wedge \chi_{\{\sigma\}}.$$

In the above computation, first, the BDD representation of all transitions that include event σ is extracted. Next, the BDD-variables used for representing the target-states and events are excluded, yielding the states in S_0 from which σ is enabled. Based on $\chi_{\mapsto_{S_0}}$, $\chi_{Q^{safe}}$, and $\chi_{Q^{\sigma}}$ (all computed earlier), the corresponding BDDs for the basic state sets are computed as below,

$$\begin{aligned}
\chi_{Q^{forbidden}} &= \exists \acute{\mathbf{b}}^{Q}, \mathbf{b}^{\Sigma} : (\chi_{\mapsto_{S_0}} \wedge \neg \chi_{Q^{safe}}), \\
\chi_{Q^{\sigma safe}} &= \chi_{Q^{\sigma}} \wedge \chi_{Q^{safe}}, \\
\chi_{Q^{\sigma}_{\mathbf{f}}} &= \chi_{Q^{\sigma safe}} \wedge \chi_{Q^{forbidden}}, \\
\chi_{Q^{\sigma}_{\mathbf{a}}} &= \chi_{Q^{\sigma safe}} \wedge \neg \chi_{Q^{\sigma}_{\mathbf{f}}}, \\
\chi_{Q^{\sigma}_{\mathbf{dc}}} &= \neg(\chi_{Q^{\sigma}_{\mathbf{a}}} \vee \chi_{Q^{\sigma}_{\mathbf{f}}}).
\end{aligned}$$

The BDD for $\chi_{Q^{forbidden}}$ represents all states that, by one transition, lead to a state not belonging to the supervisor. The BDD for $\chi_{Q^{\sigma safe}}$ represents the safe states that enable the event σ. By conjuncting the aforementioned BDDs, all safe states, where σ must be forbidden to occur are obtained, $\chi_{Q^{\sigma}_{\mathbf{f}}}$. Similarly, the other two state sets can be computed.

As stated earlier, the don't-care states will be utilized in simplifying the guard expressions. This operation is carried out directly on the BDD representation of the state set, based on the RESTRICT function by Coudert and Madre, described in [76]. Given two BDDs $\mathbf{B_1}$ and $\mathbf{B_2}$, $\mathbf{B_3} = \textsc{Restrict}(\mathbf{B_1}, \mathbf{B_2})$ simplifies $\mathbf{B_1}$, i.e., reduces the size of $\mathbf{B_1}$, under a constraint $\mathbf{B_2}$, so that $\mathbf{B_1} \wedge \mathbf{B_2} = \mathbf{B_3} \wedge \mathbf{B_2}$. Hence, $\mathbf{B_3}$ is logically equal to $\mathbf{B_1}$, on the domain defined by $\mathbf{B_2}$ and is *often* smaller than $\mathbf{B_1}$. In this manner, we can simplify the BDD representations of the state sets by constraining them under $\chi_{Q^{\sigma}_{\mathbf{dc}}}$. Consequently, the guard generated from the simplified BDD, *usually* becomes smaller. For an elaborate and verbose exposition of the symbolic computation of the basic state sets, refer to Paper 1.

4.4.2 IDD Generation

To generate the guards based on the BDDs, we need to map the Boolean variables to their corresponding states. To this end, we convert a BDD to its corresponding *integer decision diagram* (IDD) [77]. IDD is an extension to a BDD where the number of terminals is arbitrary and the domain of the variables in the graph is an arbitrary set of integers. For our purpose, we use an IDD with two terminals, 0-terminal and 1-terminal.

Using IDDs to generate guards has some advantages in comparison to BDDs: 1) they make it easier to handle and manipulate propositional formulae; 2) they exploit some of the common subexpressions in a guard yielding a more factorized and smaller formula; 3) they depict a more understandable model of the state set, since the nodes and edges represent names of the automata and states, respectively.

Each IDD-variable is associated to an automaton A_i, and each outgoing edge from node A_i represents a state in A_i, giving a maximum number of edges $|Q_{A_i}|$.

A BDD is converted to an IDD by traversing it in a top-down depth-first manner and performing the following main steps:

1. For each new BDD-node $b_i^{Q_{A_s}}$ that is reached, create an IDD rooted by A_s, denoted as idd.

2. Continue traversing until a variable $b_j^{Q_{A_t}}$ is reached where $A_t \neq A_s$.

3. Create an IDD rooted by A_t, denoted as $child$.

4. Extract the sub-BDD between $b_i^{Q_{A_s}}$ and $b_j^{Q_{A_t}}$ that represents some states of automaton A_s.

5. Add $child$ to idd's children and label the edge with $Q_{A_s}^{edge}$.

6. Repeat the procedure from step 1.

The result is correct under the assumption that the BDD has a fixed variable ordering. A pseudo algorithm of this procedure is presented in Paper 1.

4.4.3 Guard Generation

The last step of obtaining the guard is to convert the IDDs to propositional formulae. For a given IDD, a top-down depth first search is used to traverse the graph and generate its corresponding propositional formula. In Paper 1, an algorithm is presented that generates a guard based on an IDD by considering the heuristic techniques, described in Section 3.4.1, to simplify the guard. The algorithm starts from the root and visits the nodes, while generating the expression and ends at the 1-terminal. For each node in the IDD, the corresponding expressions of the edges belonging to the same level (the children of that node) are logically disjuncted and if the edges belong to different levels they are logically conjuncted. Hence, the propositional formula for the IDD in Figure 4 is

$$r \wedge ((p_1 \wedge S_1) \vee (p_2 \wedge S_2)),$$

where p_i is the corresponding expression of the edge that lead to one of A's children and S_i is the corresponding expression from the node to the 1-terminal, that is recursively computed.

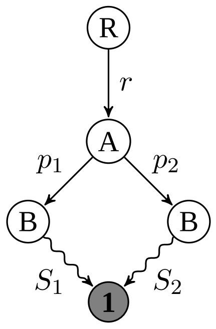


Figure 4.5: Recursive representation of an IDD.

4.4.4 Guard Reduction by Genetic Algorithms

Since a guard is generated indirectly from a BDD, the guard's size becomes very sensitive to the size of the BDD. Hence, the variable ordering of the BDD, can impact the size of the guard. Note that the smallest BDD does not necessarily yield the smallest guard. In [78], we used *genetic algorithms* (GA) to reduce the size of the generated guard by changing the variable ordering of the underlying BDD.

A GA is a search heuristic that mimics the process of natural evolution. Genetic algorithms belong to the larger class of evolutionary algorithms, which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. In a genetic algorithm, a population of strings (called *chromosomes*), which encode candidate *solutions* (called *individuals*) to an optimization problem, evolves toward better solutions. The evolution usually starts from a population of randomly generated individuals and iteratively continues by creating new generations. In each generation, the *fitness* of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm.

In the following, we briefly describe each of the operations performed during the GA.

Representation

Traditionally, GA works on binary strings of 0's and 1's. However, such encoding require a special repair operation to avoid creation of invalid solutions. Another encoding was introduced for solving Traveling Salesmen Problem [79] and later used for minimization of BDDs [80], which represents a variable ordering as an integer string of length n, where n is the number of variables in a BDD, and each integer appears in the string once. In [78], we used the latter representation and thus each individual consists of a string of variables in the BDD.

Initialization

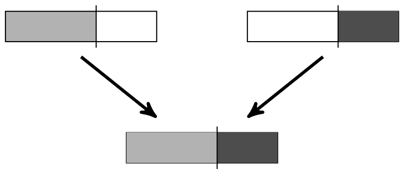
The population is initialized by generating random individuals. Starting from a randomly generated individual, the other individuals are generated by randomly permutating the chromosomes in the initial individual. If a new individual already exists in the population, it is discarded. Individuals are added until population size reaches a predefined size.

Selection

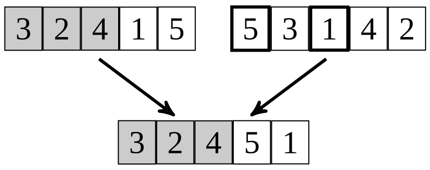
The selection of the individuals for mating pool is performed by roulette wheel selection, where each individual is chosen with a probability proportional to its fitness. As a fitness measure of an individual, the size of the guard generated using the variable ordering encoded by the individual is used. Additionally, some of the best individuals of the old generation are also included in the new generation, to ensure that the best element is never lost.

Crossover

For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the mating pool selected previously. Two parents are combined with each other using *crossover* operation to produce a “child”. New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated. In a traditional implementation of the crossover operation, a random cut point is selected, and the chromosome of the first parent is taken up to the cut point, and chromosome of the second parent is taken from cut point to the end. This, however, would produce invalid variable orderings. Instead, a crossover illustrated in Figure 4.6 is used [80, 81], where genes of the chromosome of the first parent are taken up to the cut point, while from the second parent all other missing genes are taken in the order they appear. This preserves relative order of some of the variables of both parents, and always generates valid solutions.



(a) Conventional crossover



(b) Order crossover

Figure 4.6: Crossover operation.

Mutation

Mutation helps diversifying solutions and escaping local minima. The mutation operation is carried out by swapping two genes in an individual.

Termination

The algorithm is terminated after a predefined number of iterations, or when no better individuals were produced after several consecutive iterations.

Worth to emphasise that in the GA-based approach, the goal is to find the optimal variable ordering yielding the smallest guard, rather than the smallest BDD.

4.5 Related Work

Another symbolic approach that has been applied to SCT is based on *Boolean satisfiability (SAT) solvers*. SAT solvers are programs that solve the problem of determining if the variables in a Boolean formula can be assigned in a way so that the formula is satisfied, i.e. evaluates to `true`. SAT-based techniques have been utilized in various domains, especially verification of models, and promising results have been obtained [82–85]. However, SAT-based techniques are not always efficient for synthesis [86]. In general, depending on the problem to be solved, either SAT- or BDD-based methods could be suitable, and can therefore be seen as complementary techniques.

Chapter 5

Case Studies

In this chapter, we apply the presented framework to an illustrative and an industrial example. We show how the examples can be modeled by TEFAs and how their supervisor can be computed and represented. We also briefly discuss about the BDD implementation. For more case studies and experimental results, refer to Paper 1-4.

5.1 Illustrative Example

Consider a manufacturing cell, shown in Figure 5.1, taken from [49].

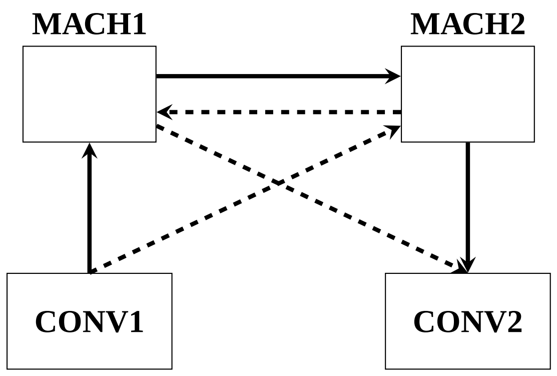


Figure 5.1: The manufacturing cell. The solid and dottel lines correspond to parts p_1 and p_2, respectively.

The manufacturing cell consists of machines **MACH1** and **MACH2**, with an input conveyor **CONV1** as an infinite source of workpieces and output conveyor **CONV2** as an infinite sink. Each machine may process two types of parts, p_1 and p_2; and each machine is liable to break down, but then may be repaired. For simplicity, the transfer of parts between machines will be absorbed as a step in machine operation. The machine TEFAs are displayed in Figure 5.2, including some given timed restrictions. For **MACH1** and **MACH2** we define two clocks c_1 and c_2, respectively, with domains $\{0,\dots,4\}$ and $\{0,\dots,5\}$. The event α_{ij} occurs when **MACH**i starts working on a p_j-part, while β_{ij} represents when **MACH**i finishes working on a p_j-part; λ_i and γ_i represent respectively the breakdown and repair of **MACH**i.

The events are categorized as follows:

$$\Sigma^f = \{\alpha_{ij} \mid i,j = 1,2\},$$
$$\Sigma^u = \{\lambda_j, \beta_{ij} \mid i,j = 1,2\},$$
$$\Sigma^c = \Sigma^f \cup \{\gamma_1, \gamma_2\}.$$

We shall impose (i) logic-based specifications, (ii) a temporal specification, and (iii) a quantitative optimality specification as follows:

(i) 1. a given part can be processed by just one machine at a time,

2. a p_1-part must be processed first by **MACH1** and then by **MACH2**,

3. a p_2-part must be processed first by **MACH2** and then by **MACH1**,

4. one p_1-part and one p_2-part must be processed in each production cycle,

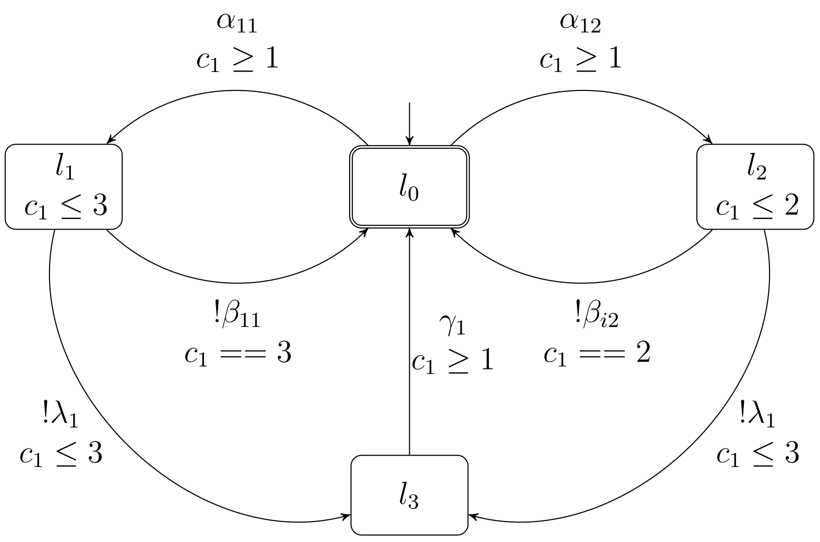
5. if both machines are down, **MACH2** is always repaired before **MACH1**;

(ii) 5. in the absence of breakdown/repair events a production cycle must be completed in at most 10 time units;

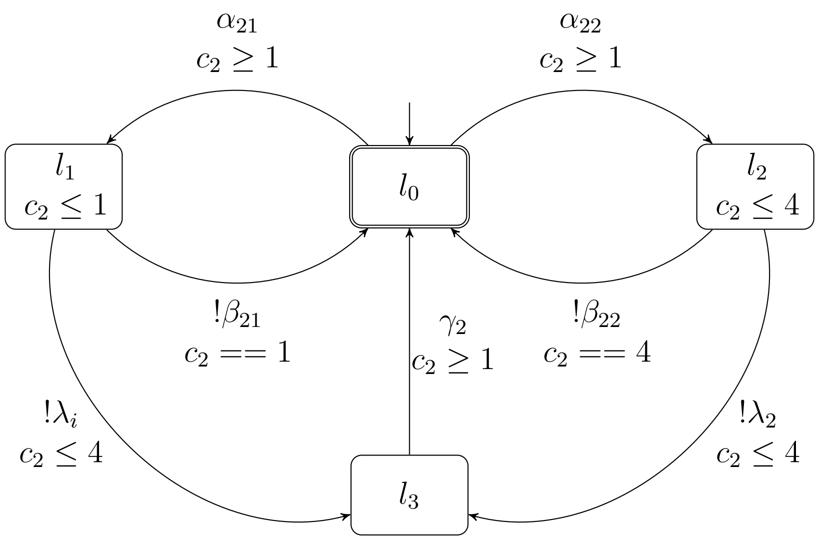
(iii) 6. subject to (ii), production cycle time is to be minimized.

We introduce two assisting variables $p1b$ and $p2b$ that are set to 1 when parts p_1 and p_2 are being processed, respectively and set to 0 when they are not processed. Thus, $p1b$ will be set to 1 on transitions including events α_{i1} and set to zero on outgoing transitions from location l_1; and similarly for variable $p2b$. The specification 1 can then be directly modeled on the machine plants, by restricting the transitions including α_{i1} with the guard $p1b == 0$; and similarly for transitions including α_{i2}. Notice that since the guards are added to transitions with controllable events, it will not cause any controllability issue. Specifications 2-6 are modeled by automata **SPEC2-SPEC6**, respectively, shown in Figure 5.3. The alphabet of each automaton is the events illustrated in each corresponding figure. It can be verified that, in fact, specification 1 is automatically enforced by specifications 2 and 3 together. We therefore only consider the composition of **SPEC2-SPEC5** as the specification of the cell. And the cell's open-loop behavior, i.e., the plant, will be the composition of **MACH1** and **MACH2**.

The system consists of 2656 reachable states, whereas 1073 states belong to the minimally restrictive supervisor. Notice that these numbers differ from

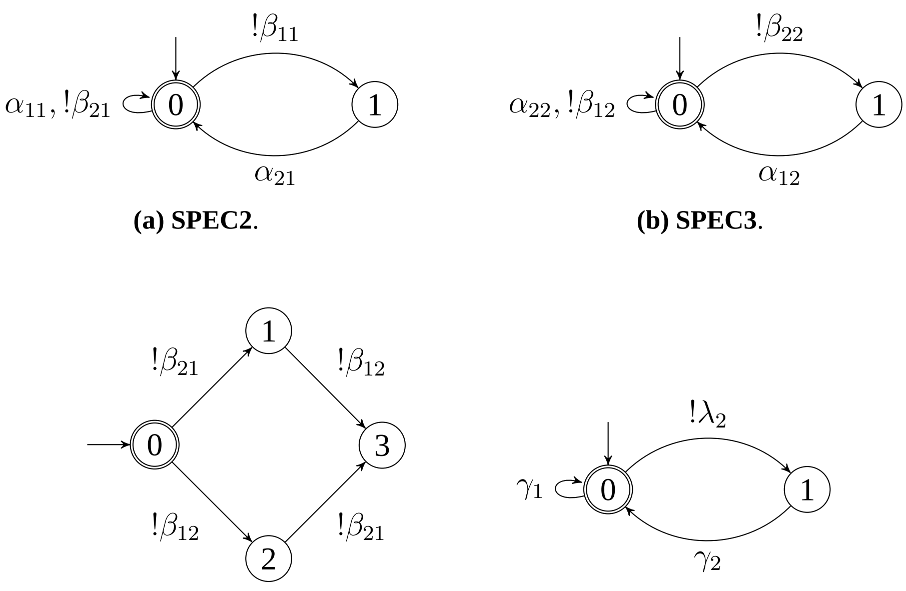


(a) MACH1.



(b) MACH2.

Figure 5.2: The TEFAs of **MACH1** and **MACH2**.



(a) SPEC2. **(b) SPEC3**. **(c) SPEC4**. **(d) SPEC5**.

Figure 5.3: The specifications of the timed manufacturing cell.

the numbers in [49]. The reason is that in our approach we implicitly let the *tick* event occur until all clocks reach their maximum values, yielding different states. On the other side, in [49], there does not exist any clocks and thus self-loop *tick* transitions will be added to states, where *tick* does not change the behavior of the model. However, the control function behavior, in the sense of Figure 3.1, of both approaches is the same.

Based on the supervisor, guards were generated for events α_{11} and α_{22}, with sizes 12 and 2, respectively. The remaining events do not require any restrictions, i.e., they are always allowed to occur without causing any problem. From an implementation point of view, an event that is always allowed or forbidden, can be directly 'hard-coded' in the plant. Hence, the plant does not need to ask the supervisor whether it is allowed to execute such an event, resulting in less communication between the plant and the supervisor. It is also worth to mention that, from a modeling perspective, knowing that some events are always allowed or forbidden to occur could be helpful, e.g., to realize what events cause problems.

As an example, the sufficient restriction on a_{22} is:

$$\mathcal{G}_{\star}^{a_{22}} : l_{\textbf{SPEC4}} == 0 \ \vee \ l_{\textbf{SPEC4}} == 1,$$

where $l_{\textbf{SPEC4}}$ is a new variable introduced to the model with domain 0,. . . ,3, representing the current location of **SPEC4**. This indicates that event a_{22} is allowed to be executed only if the system is in location 0 or 1 of **SPEC4**. Consequently, a supervisor with 1073 states has been represented by two relatively small guards. In this controlled behavior, forcing plays no role.

Figure 5.4 shows the size of intermediate BDDs in each iteration, during the reachability analysis (the RESTRICTEDFORWARD algorithm, described in 3.3.1), for both the $tick$-based approach using $tick$-EFAs and the $tick$-eliminated approach (in the sequel referred to as the TEFA-based approach) using TEFAs. It is observed that the fixed point computation based on TEFAs needs less iterations to reach a fixed point due to the fact that, in contrast to the $tick$-based approach, it does not perform iterations for the $tick$ event. Furthermore, the maximum size of the intermediate BDD in the TEFA-based approach is smaller than the $tick$-based approach. For larger examples, this could avoid state space explosion. Notice that since the TEFA-based approach starts with a set of states, the initial BDD is larger than the $tick$-based approach, which starts with a single state.

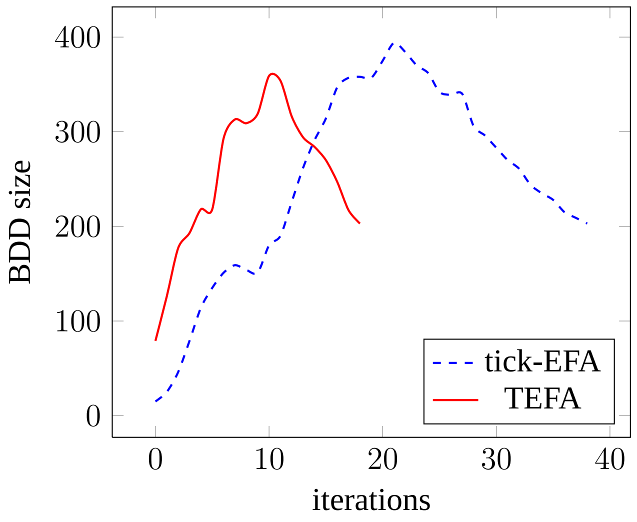


Figure 5.4: The size of intermediate BDDs in each iteration, during the reachability analysis for the timed manufacturing cell.

To address the temporal specification (ii), we first modify the models, under the stated assumption that breakdowns are absent, by removing **SPEC5** and all transitions including λ_i or γ_i events in **MACH**i. Next, we introduce a clock c_3 with domain $\{0, \ldots, 10\}$. We can now model the temporal specification by a TEFA with a single location with invariant $c_3 \leq 10$. Since c_3 evolves synchronously with c_1 and c_2, only those marked states that include a c_3 value less than 10, i.e., $\mu_3^{\mathcal{C}} \leq 10$, will be extracted. The supervisor is computed in less than a second and consists of 933 states. In [49], this specification has been modeled by an automaton with 11-$tick$ sequence all of whose states are marked. We conclude that, in the absence of breakdowns, a production cycle can indeed be forced to complete in at most 10 time units. Here, of course, the use of forcible events is essential.

Finally, to address specification (iii), based on the the marked states of the previously computed supervisor for specification (ii), the minimal value of c_3

can be extracted. Considering the state with the minimal value as the marked, we perform a new synthesis to ensure that the marked state can be reached in a controllable manner. If the synthesis does not return a supervisor, the same procedure is performed on the next minimal value of c_3. In this case, there exists a supervisor for the minimal value of c_3, having the value 7. In [49], this specification is implemented as in (ii) with successive timer sequences of $tick$-length $9, 8, \ldots$ until the synthesis algorithm returns an empty result.

5.2 Industrial Case Study

Consider a real industrial case study, taken from [87]. The goal is to design a robust and optimal controller for a plastic injection molding machine. The system to be controlled is depicted in Figure 2. It is composed of: "(1) a machine which consumes oil, (2) a reservoir containing oil, (3) an accumulator containing oil and a fixed amount of gas in order to put the oil under pressure, and (4) a pump" [87]. When the system starts, the machine consumes oil under pressure made by the accumulator. The pump can control the the level of the oil and the pressure within the accumulator to introduce additional oil into it.

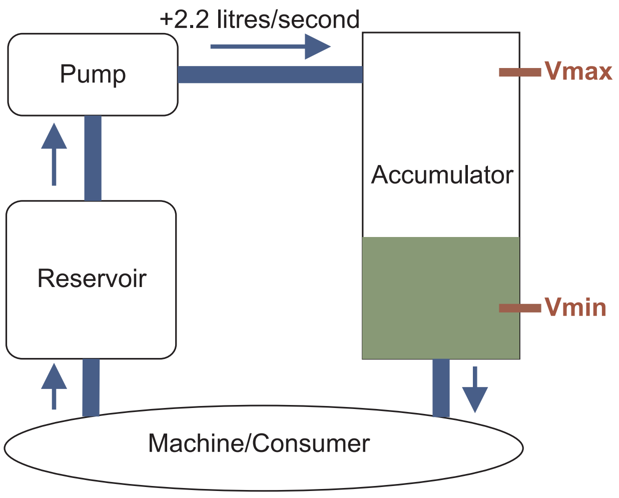


Figure 5.5: Overview of the oil pump system.

The controller must turn the pump *on* and *off* to ensure the following two main requirements [87]:

R_1: "the level of oil $v(t)$ at time t (measured in litres) into the accumulator must always stay within two safety bounds $[V_{min}; V_{max}]$, in the sequel $V_{min} = 4.9l$ and $V_{max} = 25.1l$";

R_2: "a large amount of oil in the accumulator implies a high pressure of gas in the accumulator. This requires more energy from the pump to fill in the accumulator and also speeds up the wear of the machine. It is thus desired to keep the level of oil minimal during operation, in the sense that $\int_{t=0}^{t=T} v(t)$ is minimal for a given operation period T" .

Requirement $\mathtt{R}_1$ can be seen as a qualitative specification, representing a safety property, while requirement $\mathtt{R}_2$ is a quantitative specification, representing an optimality property.

The machine consumes the oil in a cyclic manner. In each period, the machine consumes the oil by a specific rate, expressed as number of litres per second. "At time 2, the rate of the machine goes to $1.2l/s$ for two seconds. From 8 to 10 it is 1.2 again and from 10 to 12 it goes up to 2.5 (which is more than the maximal output of the pump). From 14 to 16 it is 1.7 and from 16 to 18 it is 0.5" [87]. However, there exists a *noise* of $0.1l/s$. Hence, for a specific period, if the mean consumption is cl/s, in reality the rate will lie in the interval $[c - 0.1, c + 0.1]$. This property is noted F.

The initial volume of the oil within the accumulator is assumed to be 10 l. The pump is initially *off* and when it is *on* the output rate is $2.2l/s$. It is desired that after any change of state of the pump (*on* or *off*), at least two seconds must last before the next change can happen. Furthermore, the number of times the pump can be turned on and off is restricted to two times.

Consequently, a controller is desired that, with respect to the mentioned restrictions on the pump and the measurement noise of the machine, turns the pump on and off at appropriate time points to satisfy requirement $\mathtt{R}_1$ and try to minimize the accumulated oil during each cycle (requirement $\mathtt{R}_2$). The controller should work for an arbitrary long period of time.

In [87], this system has been modeled by timed game automata [25], and the controller is synthesized using Uppaal-Tiga [62].

We transform the timed game automata in [87] to TEFAs, such that they adapt to the SCT. In contrast to the approach in [87], where around 10,000 short executions were needed to compute the optimal controller, here we compute the controller in two steps: (1) compute the minimally restrictive supervisor satisfying requirement $\mathtt{R}_1$, (2) based on this supervisor, compute a new supervisor satisfying requirement $\mathtt{R}_2$. The TEFAs of the machine, pump, and scheduler are shown in Figure 3, 4, and 5, respectively. We briefly describe the TEFAs and explain how they have been modeled in the context of SCT. Since the TEFAs are quite similar to the models in [87], for a detailed description of the TEFAs, the reader is referred to [87].

The machine and the pump TEFAs are considered as plant. The specification is modeled by the scheduler and the explicitly forbidden location bad. The events

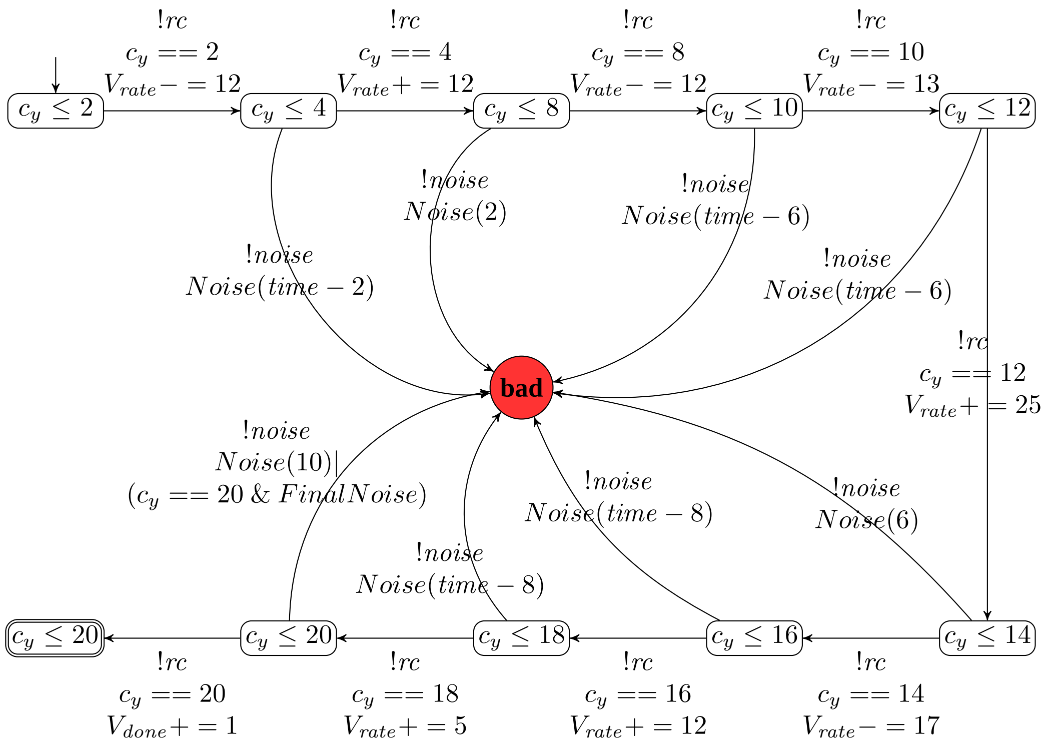
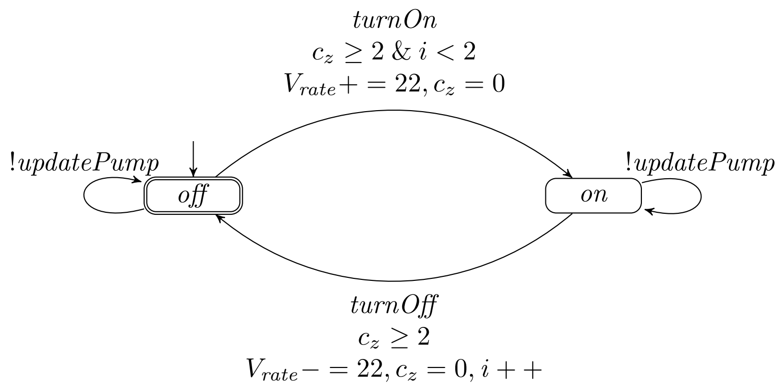


Figure 5.6: The TEFA of the cyclic consumption of the machine.



$turnOn$
$c_z \geq 2 \ \& \ i < 2$
$V_{rate} += 22, c_z = 0$
$!updatePump$
off
on
$!updatePump$
$turnOff$
$c_z \geq 2$
$V_{rate} -= 22, c_z = 0, i++$

Figure 5.7: The TEFA of the pump.

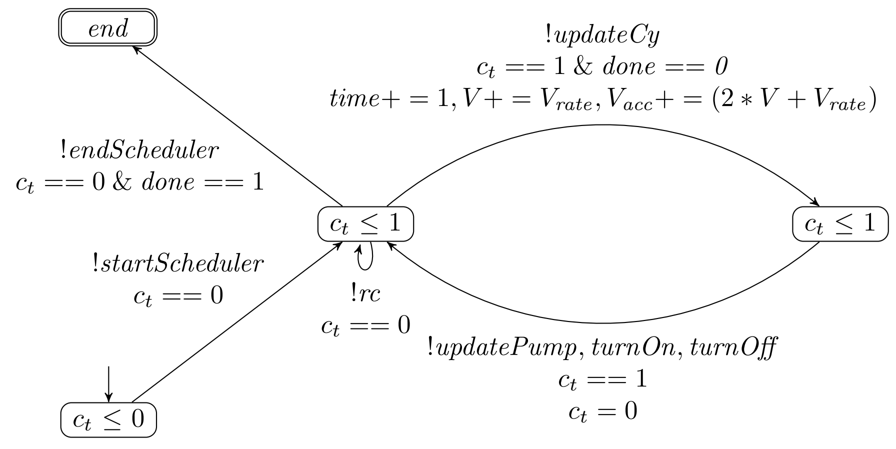


Figure 5.8: The TEFA of the scheduler.

are categorized as follows:

$$\begin{aligned}\Sigma^f &= \{turnOn, turnOff, startScheduler, endScheduler\},\\ \Sigma^c &= \{turnOn, turnOff\},\\ \Sigma^u &= \Sigma\backslash\Sigma^c.\end{aligned}$$

The alphabet of each TEFA is the set of events depicted in each corresponding figure. The model consists of the following variables clocks:

V: a variable with domain $\{0, \ldots, 255\}$, representing the current volume of oil,

V_{rate}: a variable with domain $\{-25, \ldots, 25\}$, representing the rate that V evolves,

V_{acc}: a variable with domain $\{0, \ldots, 2047\}$, representing the accumulated volume of oil,

$time$: a variable with domain $\{0, \ldots, 31\}$, representing the global time since the beginning of the cycle,

i: a variable with domain $\{0, \ldots, 2\}$, representing the number of timed the pump has been turned on and off,

$done$: a variable with domain $\{0, \ldots, 1\}$, representing when a cycle is finished,

c_y: a clock with domain $\{0, \ldots, 21\}$,

c_z: a clock with domain $\{0, \ldots, 21\}$,

c_t: a clock with domain $\{0, \ldots, 2\}$.

We have considered a precision of $0.1l$ and thus, to use integers, the value of the volume is multiplied by 10.

The transitions of the TEFA, except the ingoing transitions to location bad, of the machine follow easily from the given cyclic definition of the consumption of the machine. The guard $Noise(s)$ will be satisfied if the current volume exceeds the boundary of V_{min} and V_{max}, i.e., 4.9 and 25.1, due to fluctuations of the consumption:

$$Noise(s) = (V - s < 50) \mid (V + s) > 250.$$

The guard $FinalNoise$ checks the same but for the volume obtained at the end of cycle and against the interval represented by $V1F$ and $V2F$ that are two variables with equal domains $\{0, \ldots, 255\}$:

$$FinalNoise = (V - 10 < V1F) \mid (V + 10) > V2F.$$

Notice that $Noise$ and $FinalNoise$ are modeling the property F.

The scheduler is used to get the correct behavior of the model: the variables $time$, V, and V_{acc} should be updated after each rate change, i.e., after each transition, where V_{rate} gets updated.

The compositional model will correspond to a single cycle. However, as stated earlier, the goal is to have a controller that works properly for any number of cycles. To extend the approach to a number of cycles, we follow the same technique as [87]: "find some interval $I_1 = [V_1, V_2] \subseteq [4.9; 25.1]$ such that:

(i) I_1 is stable: from all initial volume $V_0 \in I_1$, there exists a strategy for the controller to ensure that whatever the fluctuations on the consumption, the value of the volume is always between 5 l and 25 l and the volume at the end of the cycle is within interval $I_2 = [V1F, V2F]$, where $V1F = V_1 + 0.4$ and $V2F = V_2 - 0.4$, and 0.4 is a margin parameter considered to ensure robustness,

(ii) I_1 is optimal among stable intervals: the worst accumulated volume of the solutions of I_1 is minimal".

We perform each step separately.

We start by satisfying property (i). As it can be observed, the objective of this problem is to find some proper values for variables, which is slightly different from the objectives usually defined in the SCT context. To handle this, we use a trick: let the initial values of the variables V, $V1F$, and $V2F$ be the entire corresponding domains. Fortunately, this can be handled easily by BDDs. In particular, by starting with all possible values of V we compute several supervisors in parallel (this is the main advantage of symbolic computations). However,

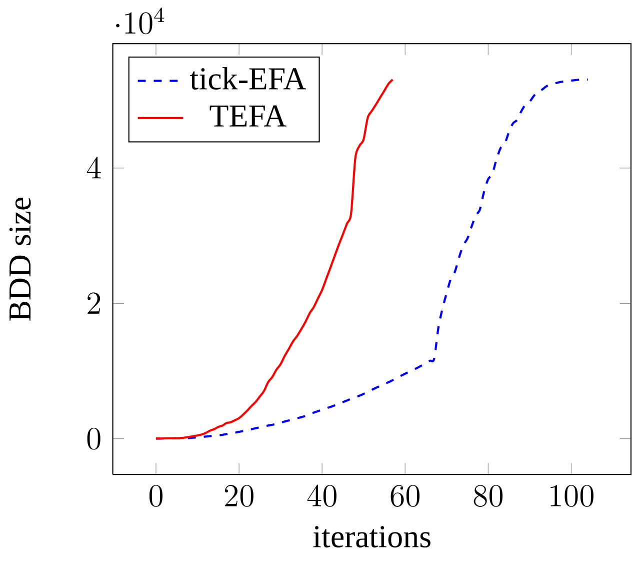


Figure 5.9: The size of intermediate BDDs in each iteration, during the reachability analysis for the oil pump systems.

to keep track of the corresponding initial values of V for the marked states, we construct the following BDD that will represent the initial states:

$$\bigvee_{i=1}^{255} \mathbf{b}_V^{\mathcal{V}} \leftrightarrow \theta(i) \wedge \mathbf{b}_{V0}^{\mathcal{V}} \leftrightarrow \theta(i),$$

where $V0$ is a new variable with domain $\{0, \ldots, 255\}$. The value of variable $V0$ can be considered as the identity of the states that will be followed during the fixed point computations. Consequently, the synthesized supervisor will only contain those initial values where a marked state can be reached, which represents the interval I_1. The minimally restrictive supervisor was computed in 2 minutes and 13 seconds and consists of 7,846,603 states. Figure 6 shows the size of intermediate BDDs in each iteration, during the reachability analysis, for the $tick$-based approach and the TEFA-based approach. It can be observed that, in both cases, due to the special treatment of the variables, the size of the BDDs grows exponentially. Furthermore, we can see that the TEFA-based approach has reached a fixed point much earlier than the $tick$-based approach. However, from a BDD size point of view, eliminating the $tick$ event did not gain so much.

Based on the computed supervisor, we perform the optimization, i.e., property (ii). The main idea is to select a subset of the reached marked states and perform a further backward reachability. Note that each marked state of the supervisor, now includes the values of the variables $V0$ and V_{acc}. Hence, among the marked states, fixing $V0$ to a specific value v, we obtain all values of V_{acc}, which can be reached safely by starting with volume v. By a simple BDD operation, we can extract the minimal value of V_{acc} among all marked states with

$V0 = v$. By performing this on all values of $v \in V_0$, we get a BDD, representing the states that include $(v, \min\{V_{acc}^v)\}$. Among these states, we extract an interval $I_1 = [v_1, v_2]$, where the maximum value of $\min\{V_{acc}^v)\}$ among all values in I_1 is minimal compared to other possible intervals. We consider the states that contain the interval I_1 as the new marked states. Based on the computed marked states, by performing a backward reachability on the earlier computed supervisor, we get a new supervisor with 2,431,982 states that was computed in 58 seconds. The corresponding interval I_1 of this supervisor is $[51, 100]$. The time points for turning the pump on and off can be obtained by checking the $time$ variable in the corresponding guards of events $turnOn$ and $turnOff$. Due to many different configurations the system can be in, the guards become very large, and not tractable for the designers. They can though be implemented in a controller directly. Basically, the guards have the following format:

$$(V0 == v \ \wedge \ time == t \ \wedge \ \ldots) \ \vee \ \ldots .$$

Hence, for each event $turnOn$ or $turnOff$, it can be deduced at what time the pump should be turned on or off, respectively. However, since the guards are large, to identify the above statement among hundreds of terms is not easy. Nonetheless, we can still use the BDD representing the allowed state set (described in Section 3.4.1), to achieve this information. Table 1 shows the time points at which the pump should be turned on and off for different initial volumes in the interval I_1. In the table, $time_i^{\text{on}}$ and $time_i^{\text{off}}$, represents the time point the pump should be turned on and off, respectively, for $i = 1, 2$.

Table 5.1: The time points at which the pump should be turned on and off for different initial volumes in the interval $I_1 = [5.1, 10.0]$.

$V0$	$time_1^{\text{on}}$ / $time_1^{\text{off}}$	$time_2^{\text{on}}$ / $time_2^{\text{off}}$
$[5.1, 5.3]$	2 / 4	9 / 15
$[5.3, 6.4]$	2 / 4	9 / 14
$[6.4, 6.7]$	3 / 5	9 / 14
$[6.7, 7.5]$	3 / 5	10 / 15
$[7.5, 7.7]$	3 / 5	10 / 14
$[7.7, 8.5]$	8 / 12	14 / 16
$[8.5, 8.8]$	8 / 12	15 / 17
$[8.8, 9.0]$	8 / 11	14 / 17
$[9.0, 9.7]$	9 / 12	14 / 17
$[9.7, 10.0]$	9 / 12	14 / 16

These results conform with results obtained in [87].

5.3 Implementation Remarks

The entire framework, discussed in the thesis, has been implemented and integrated in Supremica [29] which uses *JavaBDD* [88] as the BDD package. The experiments were carried out on a standard PC (Intel Core 2 Quad CPU @ 2.4 GHz and 3 GB RAM) running Windows 7.

Chapter 6

Summary of Appended Papers

Part II of the thesis consists of four 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

S. Miremadi, K. Åkesson and B. Lennartson. Symbolic computation of reduced guards in supervisory control. *IEEE Transactions on Automation Science and Engineering*, October 2011.

The main focus in this paper is to, based on DFAs, show how to generate guards representing the supervisor. Based on the supervisor, for each controllable event σ, the states where σ can be enabled in the composed model is divided into two basic sets: the states from which σ must be enabled to end up in the supervisor, and the states from which σ must be forbidden to be executed to not end up in an undesired state. The basic state sets are symbolically computed using BDDs. The remaining states are identified as don't-care states that are used in a BDD operator to reduce the size of the BDDs representing the basic state sets, which could lead to smaller guards. To obtain tractable guard expressions, by exploiting the structure of the given models, some heuristic techniques are applied to the guards.

Paper 2

S. Miremadi, B. Lennartson and K. Åkesson. A BDD-based approach for modeling plant and supervisor by extended finite automata. *IEEE Transactions on Control Systems Technology*, November 2012.

This paper extends the approach in Paper 1, by performing the guard generation on EFAs, FAs augmented by discrete variables. Modeling systems using EFAs will typically yield more compact models by hiding some of the states of the system in variables. The main contribution of this paper was to show how EFAs and their full synchronous composition can be symbolically computed by BDDs representing the corresponding DFAs of the EFAs. Based on the symbolic representations, the guards can be generated according to Paper 1. The generated guards can then be attached to the original models, yielding a modular supervisor.

Paper 3

S. Miremadi, Z. Fei, K. Åkesson and B. Lennartson. Symbolic representation and computation of timed discrete event systems. Submitted to *IEEE Transactions on Automation Science and Engineering*, 2012.

This paper considers time in EFAs, by presenting timed EFAs. It is shown how TEFAs can be transformed to EFAs by treating the clocks as regular variables and introducing the $tick$ event to the model, representing the time evolution. However, $tick$ models suffer from a major problem: the state size is very sensitive to the clock frequency. To tackle this problem, we proposed a method to eliminate the $tick$ events while still obtain the same behavior. The main contribution was to show how $tick$-eliminated models can be symbolically represented by BDDs. It was shown that, in this way, smaller intermediate BDDs and less iterations in the fixed point computations can be obtained. We showed how SCT can be applied to the symbolic representations by considering the $tick$ event as an uncontrollable event.

Paper 4

S. Miremadi, Z. Fei, K. Åkesson and B. Lennartson. Symbolic supervisory control of timed discrete event systems. Submitted to *IEEE Transactions on Control Systems Technology*, 2012.

In Paper 3, we assumed that the $tick$ event is uncontrollable. In Paper 4, in the context of SCT for TDES [49], we treat the $tick$ event in a special manner. As in [49], the concept of forcible events are introduced that can preempt the $tick$ event. The main contribution of this paper is to show how the synthesis, especially controllability, can be symbolically performed on the $tick$-eliminated models, presented in Paper 3. Papers 1-4 can be considered as a framework, where one is able to model a DES or TDES as EFAs or TEFAs, and symbolically compute its supervisor based on the SCT, and finally generate guards representing the supervisor and attach them to the original models.

Chapter 7

Conclusions and Future Research

As discussed in the thesis, supervisory control theory is a model-based theoretical framework for computing a control function, i.e., supervisor, that restricts a given plant towards a given specification only when it is necessary. In Chapter 1, we pointed out three challenges that exist when SCT is used:

(i) Most of the existing work on SCT, has been carried out on untimed DES for analyzing the qualitative properties of the systems. However, in most of the real-time applications, the correct behavior can only be obtained by taking time into consideration. Also, including time in the models, opens the possibility of performing quantitative analysis such as time optimization.

(ii) As discussed, the number of states of a system consisting of a number of components grows exponentially as the number of components increases. For many of the industrial applications that consist of a large number of components, this leads to state space explosion, that is the number of states cannot be represented in the hardware.

(iii) For industrial applications, typically the synthesized supervisor consists of a large number of states. Representing the supervisor could then be challengeable, both from a modeling and implementation perspective.

In this thesis, we tackled the above issues. To meet Challenge (i), we modeled the systems by TEFAs that include a set of discrete-valued clocks. The SCT for TEFAs was defined based on their corresponding $tick$-EFAs as in [49], where the clocks were considered as regular variables and the time semantics was implemented by the $tick$ event, treated in a special manner. We showed that the $tick$ models suffer from a major problem: the state size is very sensitive to the clock frequency. To tackle this problem, we proposed a method to eliminate the $tick$ events while still obtain the same behavior.

To tackle Challenge (ii), all computations were performed symbolically using BDDs. Essentially, based on a given set of TEFAs, the supervisor was computed symbolically using BDDs. We showed that the symbolic implementation of the $tick$-eliminated models result in smaller intermediate BDDs and less iterations in the fixed point computations. For some applications, this could resolve the state space explosion, caused by time.

Finally, to tackle Challenge (iii), the supervisor was represented in a modular fashion by extracting constraining guards and attaching them to the original models. In this way,

1. the designers will remain in the modular scope, which makes it possible to easily perform modifications on the resulting supervisor, e.g., changing the specification,

2. it becomes possible implement the supervisor in a modular manner, which could especially be beneficial for hierarchical approaches,

3. the final representation will be closer to the ones typically used in the industry for implementing a controller.

The guards were generated based on some categorized states of the supervisor, referred to as the basic state sets. It was shown how the basic state sets can be symbolically computed using BDDs. Furthermore, different techniques were proposed to simplify the guards. Notice that the entire procedure can also be applied to untimed DES modeled by EFAs. A process overview of the entire framework is illustrated in Figure 7.1.

The framework has been implemented and verified in the supervisory tool Supremica, and has been applied to different examples and industrial case studies, some discussed in Chapter 5.

There are some possible directions for future research. In this work, the main emphasis has been on representing the systems symbolically, rather than developing efficient synthesis algorithms. It is indeed possible to improve the efficiency of the supervisory synthesis, e.g., by utilizing partitioning techniques in the BDD computations such as [89, 90]. Furthermore, even though some techniques have been proposed to simplify the guards, still for some applications, the guards may become complicated. Essentially, it is possible to simplify the guards more by utilizing the behavioral structure of the models.

Analyzing timed systems, a missing piece in this thesis, is an approach to automatically perform time optimization on the TEFAs. The interesting point about time optimization on TEFAs is the existence of uncontrollable events that may lead to several optimal solutions. In particular, disregarding the uncontrollable events, there may exist a path from the initial state to a marked state that takes minimal time to reach. However, if there exists an outgoing uncontrollable event from a state in the optimal path, which could not be restricted by the supervisor, the system can end up in a state not belonging to the optimal path anymore. In such a case, we may desire a new minimal path from the new state

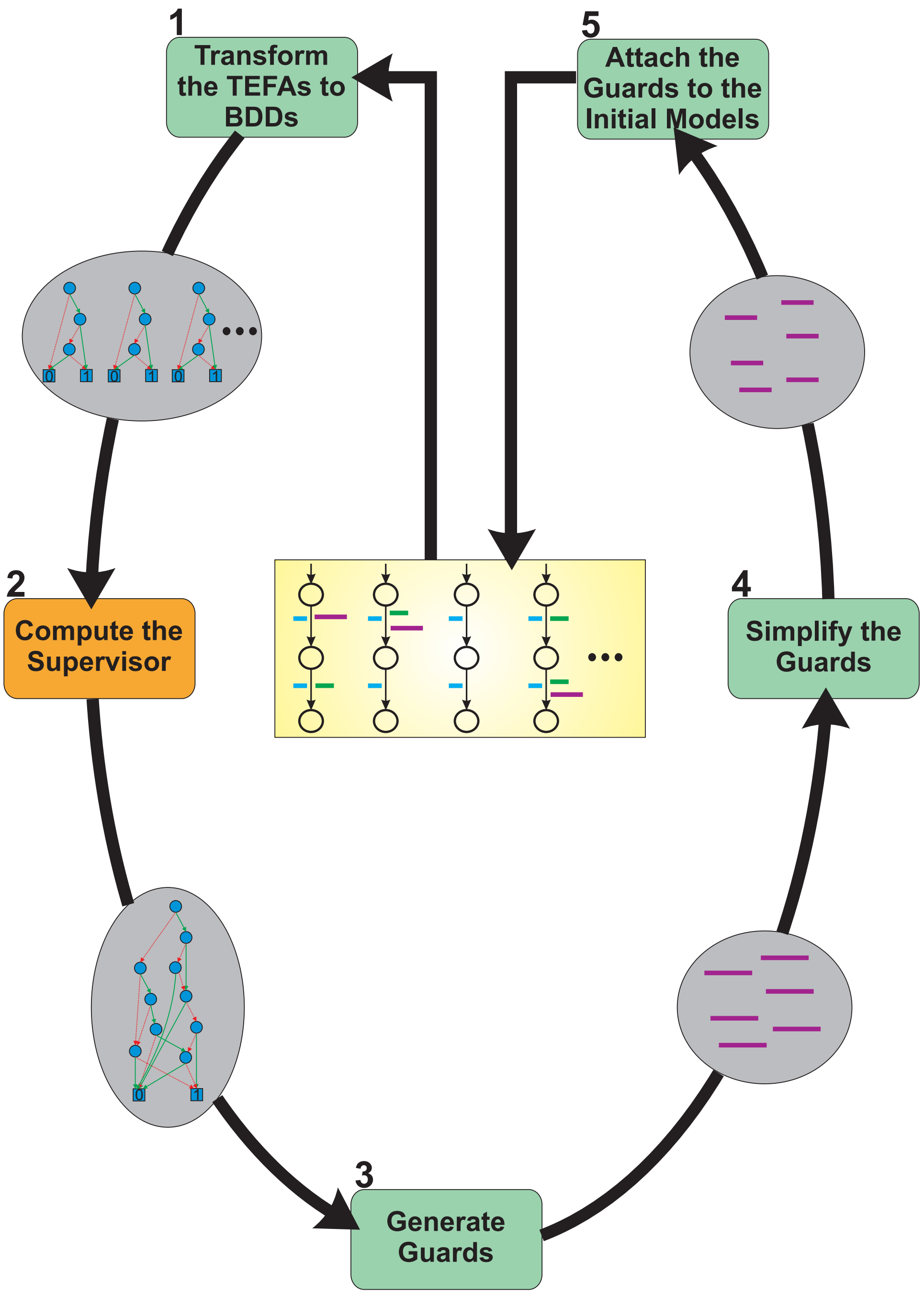

1
Transform the TEFAs to BDDs
5
Attach the Guards to the Initial Models
2
Compute the Supervisor
4
Simplify the Guards
3
Generate Guards
0
1

Figure 7.1: Process overview of the approach.

to a marked state. A possible way for solving this problem could be to, first, compute the minimal time from each state to a marked state. This can be achieved by performing a backward reachability computation from all the marked states including all possible values of the global clock. Second, based on the minimal times, a one-step lookahead strategy could be computed for each state, indicating the event(s) that will finally yield the minimal time.

Bibliography

[1] A. Wolfe, “For Intel, it’s a case of FPU all over again,” *EE Times*, 1997.

[2] A. Mishkin, J. Morrison, T. Nguyen, H. Stone, B. Cooper, and B. Wilcox, “Experiences with operations and autonomy of the Mars Pathfinder Microrover,” in *IEEE Aerospace Conference*, vol. 2, 1998, pp. 337–351.

[3] J. Rawlinson, “Report on the Therac-25,” in *OCTRF/OCI Physicists Meeting*, Kingston, Ontario, 1987.

[4] V. D’Silva, D. Kroening, and G. Weissenbacher, “A survey of automated techniques for formal software verification,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1165–1178, Jul. 2008.

[5] C. Kern and M. R. Greenstreet, “Formal verification in hardware design: a survey,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 4, no. 2, pp. 123–193, Apr. 1999.

[6] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, “UPPAAL – a tool suite for automatic verification of real-time systems,” *Lecture Notes in Computer Science*, vol. 1066, no. 1996, pp. 232–243, 1996.

[7] S. Yovine, “KRONOS: a verification tool for real-time systems,” *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-2, pp. 123–133, 1997.

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

[9] B. A. Brandin and F. E. Charbonnier, “The supervisory control of the automated manufacturing system of the AIP,” in *Proceedings of the 4th International Conference on Computer Integrated Manufacturing and Automation Technology*, Oct. 1994, pp. 319–324.

[10] V. Chandra, Z. Huang, and R. Kumar, “Automated control synthesis for an assembly line using discrete event system control theory,” *IEEE Trans. on Systems, Man and Cybernetics*, vol. 33, no. 2, pp. 284–289, 2003.

[11] A. Giua and C. Seatzu, “Supervisory control of railway networks with Petri nets,” in *Proceedings of the 40th IEEE Conference on Decision and Control*, vol. 5, 2001, 5004–5009 vol.5.

[12] M. A. Jafari, H. Darabi, T. O. Boucher, and A. Amini, “A distributed discrete event dynamic model for supply chain of business enterprises,” in *Proceedings of the 6th International Workshop on Discrete Event Systems, WODES’02*, 2002, pp. 279–285.

[13] L. Feng, W. M. Wonham, and P. S. Thiagarajan, “Designing communicating transaction processes by supervisory control theory,” *Form. Methods Syst. Des.*, vol. 30, no. 2, pp. 117–141, 2007.

[14] M. Seidl, “Systematic controller design to drive high-load call centers,” *IEEE Transactions on Control Systems Technology*, vol. 14, no. 2, pp. 216–223, Mar. 2006.

[15] 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.

[16] L. Feng and W. M. Wonham, “TCT: A computation tool for supervisory control synthesis,” in *Proceedings of the 8th international Workshop on Discrete Event Systems, WODES’06*, 2006, pp. 388–389.

[17] B. A. Brandin and W. M. Wonham, “Supervisory control of timed discrete-event systems,” *IEEE Transactions on Automatic Control*, vol. 39, no. 2, pp. 329–342, 1994.

[18] H. Chen and H. Li, “Maximally permissive state feedback logic for controlled time Petri nets,” in *Proceedings of the 1997 American Control Conference*, vol. 4, American Autom. Control Council, 1997, pp. 2359–2363.

[19] A. Saadatpoor, “Timed state tree structures: superviory control and fault diagnosis,” Ph.D. dissertation, University of Toronto, 2009.

[20] H. Wong-Toi and G. Hoffmann, “The control of dense real-time discrete event systems,” in *Proceedings of the 30th IEEE Conference on Decision and Control*, IEEE, 1991, pp. 1527–1528.

[21] R. Alur and D. L. Dill, “A theory of timed automata,” *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, Apr. 1994.

[22] E. Asarin, O. Maler, and A. Pnueli, “Symbolic controller synthesis for discrete and timed systems,” *Hybrid Systems II - Lecture Notes in Computer Science*, vol. 999, pp. 1–20, 1995.

[23] P. Niebert, S. Tripakis, and S. Yovine, “Minimum-time reachability for timed automata,” in *8th IEEE Mediterranean Conf. on Control and Automation*, 2000.

[24] T. Brihaye, T. A. Henzinger, V. S. Prabhu, and J.-F. Raskin, “Minimum-time reachability in timed games,” in *34th International Colloquium*, Springer Berlin Heidelberg, 2007, pp. 825–837.

[25] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime, “Efficient on-the-fly algorithms for the analysis of timed games,” in *Proceedings of the 16th International Conference on Concurrency Theory*, 2005, pp. 66–80.

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

[27] 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.

[28] Supremica, *`WWW.SUPREMICA.ORG`. THE OFFICIAL WEBSITE FOR THE SUPREMICA PROJECT*, 2004.

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

[30] 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.

[31] A. Arnold and J. Plaice, *Finite transition systems: semantics of communicating systems*. Hertfordshire, UK, UK: Prentice Hall International (UK) Ltd., 1994.

[32] R. P. Kurshan, *Computer-aided verification of coordinating processes: the automata-theoretic approach*. Princeton, NJ, USA: Princeton University Press, 1994.

[33] A. Giua, “Petri Nets as discrete event models for supervisory control,” PhD thesis, Rensselaer Polytechnic Institute, Troy, New York, USA, Jul. 1992.

[34] J. Bergstra and J. Klop, “Process algebra for synchronous communication,” *Information and control*, vol. 60, no. 1-3, pp. 109–137, 1984.

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

[36] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems*. New York, NY, USA: Springer-Verlag New York, Inc., 1992.

[37] G. D. Plotkin, “A structural approach to operational semantics,” Århus University, Tech. Rep., Sep. 1981.

[38] C. A. R. Hoare, “Communicating sequential processes,” *Communications of the ACM*, vol. 21, no. 8, pp. 666–667, 1978.

[39] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008, p. 975.

[40] 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.

[41] J. Bengtsson and W. Yi, “Timed automata: Semantics, algorithms and tools,” *Lectures on Concurrency and Petri Nets*, vol. 3098/2004, pp. 87–124, 2004.

[42] A. Dubey, “A discussion on supervisory control theory in real-time discrete event systems,” Institute for Software Integrated Systems, Tech. Rep., 2009, p. 9.

[43] R. Alur and T. Henzinger, “Real-time logics: complexity and expressiveness,” in *Proceedings of 5th Annual IEEE Symposium on Logic in Computer Science*, IEEE Comput. Soc. Press, 1990, pp. 390–401.

[44] T. A. Henzinger, Z. Manna, and A. Pnueli, “What good are digital clocks?,” in *19th International Colloquium on Automata, Languages and Programming*, 1992, pp. 545–558.

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

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

[47] L. Ouedraogo, R. Kumar, R. Malik, and K. Åkesson, “Nonblocking and safe control of discrete-event systems modeled as extended finite automata,” *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 3, pp. 560–569, Jul. 2011.

[48] G. Cengic, “A control software development method using IEC 61499 function blocks , simulation and formal verification,” *Development*, pp. 22–27, 2008.

[49] B. A. Brandin and W. M. Wonham, “The supervisory control of timed DES,” *IEEE Transactions on Automatic Control*, vol. 39, no. 2, pp. 329–342, 1994.

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

[51] W. M. Wonham and P. Ramadge, “Modular supervisory control of discrete-event systems,” *Mathematics of Control Signals and Systems*, vol. 1, no. 1, pp. 13–30, 1988.

[52] M. H. de Queiroz and J. E. R. Cury, “Modular supervisory control of large scale discrete event systems,” in *Discrete Event Systems, Analysis and Control*, R. Boel and G. Stremersch, Eds., Kluwer, 2000, pp. 103–110.

[53] K. Åkesson, H. Flordal, and M. Fabian, “Exploiting modularity for synthesis and verification of supervisors,” in *15th IFAC World Congress*, Barcelona, Spain, 2002.

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

[55] S. Mohajerani, R. Malik, S. Ware, and M. Fabian, “Compositional synthesis of discrete event systems using synthesis abstraction,” in *Chinese Control and Decision Conference CCDC*, IEEE, May 2011, pp. 1549–1554.

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

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

[58] A. Hellgren, M. Fabian, and B. Lennartson, “Synchronized execution of discrete event models using sequential function charts,” in *Proceedings of the 38th IEEE Conference on Decision and Control*, Phoenix AZ, USA, 1999, pp. 2237–2242.

[59] A. Hellgren, B. Lennartson, and M. Fabian, “Modelling and PLC-based implementation of modular supervisory control,” in *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*, 2002, pp. 371–376.

[60] S. Miremadi, K. Åkesson, and B. Lennartson, “Symbolic computation of reduced guards in supervisory control,” *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 4, pp. 754–765, 2011.

[61] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis, “Controller symthesis for timed automata,” in *In Proceedings of IFAC Symposium on System Structure and Control*, 1998, pp. 469–474.

[62] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime, “Uppaal-tiga: Time for playing games!,” in *Proceedings of the 19th international Conference on Computer Aided Verification*, 2007,

[63] P. Gohari and W. M. Wonham, “On the complexity of supervisory control design in the RW framework.,” *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, vol. 30, no. 5, pp. 643–52, Jan. 2000.

[64] K. Rohloff and S. Lafortune, “On the computational complexity of the verification of modular discrete-event systems,” in *Proceedings of the 41st IEEE Conference on Decision and Control*, vol. 1, IEEE, 2002, 16–21 vol.1.

[65] G. Hoffmann and H. Wong-Toi, “Symbolic synthesis of supervisory controllers,” in *1992 American Control Conference*, Chicago, IL, USA, 1992, pp. 2789–2793.

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

[67] K. Schmidt, H. Marchand, and B. Gaudin, “Modular and decentralized supervisory control of concurrent discrete event systems using reduced system models,” in *Proceedings of the 8th International Workshop on Discrete Event Systems, WODES’06*, Jul. 2006, pp. 149–154.

[68] 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.*, Jun. 1990, pp. 428–439.

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

[70] C. E. Shannon, “A mathematical theory of communication,” *The Bell System Technical Journal*, vol. 27, pp. 379–423, 625656–, 1948.

[71] 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.

[72] R. Bryant, “Graph-based algorithms for boolean function manipulation,” *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

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

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

[75] 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.

[76] O. Coudert and J. C. Madre, “A unified framework for the formal verification of sequential circuits,” *1990 IEEE International Conference on Computer-Aided Design, 1990. ICCAD-90. Digest of Technical Papers.*, pp. 126–129, Nov. 1990.

[77] J. Gunnarsson, “Symbolic methods and tools for discrete event dynamic systems,” PhD thesis, Electrical Engineering, Linköping University, Linköping, Sweden, 1997.

[78] S. Miremadi and A. Voronov, “Symbolic reduction of guards in supervisory control using genetic algorithms,” Chalmers University of Technology, Gothenburg, Sweden, Tech. Rep., 2012, p. 7.

[79] L. D. Whitley, T. Starkweather, and D. Fuquay, “Scheduling problems and traveling salesmen: The genetic edge recombination operator,” in *Proceedings of the 3rd International Conference on Genetic Algorithms*, 1989, pp. 133–140.

[80] R. Drechsler, “Genetic algorithm for variable ordering of OBDDs,” in *IEE Proceedings of Computers and Digital Techniques*, 1996, pp. 364–368.

[81] D. Goldberg and R. Lingle, “Alleles, loci, and the traveling salesman problem,” in *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, USA, 1985, pp. 156–159.

[82] N. Amla, R. Kurshan, K. L. McMillan, and R. Medel, “Experimental analysis of different techniques for bounded model checking,” in *Proceedings of the 9th international conference on Tools and algorithms for the construction and analysis of systems, TACAS’03*, Berlin, Heidelberg: Springer-Verlag, 2003, pp. 34–48.

[83] A. Biere, E. Clarke, R. Raimi, and Y. Zhu, “Verifying safety properties of a powerPC microprocessor using symbolic model checking without BDDs,” in *In Proc. 11 th Int. Conf. on Computer Aided Verification*, Springer-Verlag, 1999, pp. 60–71.

[84] P. Bjesse, T. Leonard, and A. Mokkedem, “Finding bugs in an Alpha microprocessor using satisfiability solvers,” in *Proceedings of the 13th International Conference on Computer Aided Verification, CAV’01*, London, UK: Springer-Verlag, 2001, pp. 454–464.

[85] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Y. Vardi, "Benefits of bounded model checking at an industrial setting," in *Proceedings of the 13th International Conference on Computer Aided Verification, CAV'01*, London, UK: Springer-Verlag, 2001, pp. 436–453.

[86] A. Voronov and K. Åkesson, "Supervisory control using satisfiability solvers," in *9th International Workshop on Discrete Event Systems, 2008.*, May 2008, pp. 81–86.

[87] F. Cassez, J. J. Jessen, K. G. Larsen, J.-F. Raskin, and P.-A. Reynier, "Automatic synthesis of robust and optimal controllers — an industrial case study," in *Proceedings of the 12th International Conference on Hybrid Systems: Computation and Control*, 2009, pp. 90–104.

[88] *JavaBDD*. [Online]. Available: `javabdd.sourceforge.net`.

[89] B. J.R., C. D, and D. E. Long, "Symbolic model cheking with partitioned transition relations," in *A. Halaas and P.B. Denyer, editors, International Conference on Very Large Scale Integration*, Aug. 1991, pp. 49–58.

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

Part II

Appended Papers

Paper 1

Symbolic Computation of Reduced Guards in Supervisory Control

S. Miremadi, K. Åkesson and B. Lennartson

IEEE Transactions on Automation Science and Engineering, October 2011

Comment: The layout of this paper has been reformatted in order to comply with the rest of the thesis.

Paper 2

A BDD-based Approach for Modeling Plant and Supervisor by Extended Finite Automata

S. Miremadi, B. Lennartson and K. Åkesson

IEEE Transactions on Control Systems Technology, November 2012

Comment: The layout of this paper has been reformatted in order to comply with the rest of the thesis.

Paper 3

Symbolic Representation and Computation of Timed Discrete Event Systems

S. Miremadi, Z. Fei, K. Åkesson and B. Lennartson

submitted to
IEEE Transactions on Automation Science and Engineering, 2012

Comment: The layout of this paper has been reformatted in order to comply with the rest of the thesis.

Paper 4

Symbolic Supervisory Control of Timed Discrete Event Systems

S. Miremadi, Z. Fei, K. Åkesson and B. Lennartson

submitted to
IEEE Transactions on Control Systems Technology, 2012

Comment: The layout of this paper has been reformatted in order to comply with the rest of the thesis.