Model-based supervisory control synthesis of cyber-physical systems

MODEL-BASED SUPERVISORY CONTROL SYNTHESIS OF CYBER-PHYSICAL SYSTEMS

AIDA RASHIDINEJAD

Model-Based Supervisory Control Synthesis of Cyber-Physical Systems

Aida Rashidinejad

Department of Mechanical Engineering
EINDHOVEN UNIVERSITY OF TECHNOLOGY
Eindhoven, The Netherlands. 2021

The work described in this thesis was carried out at the Eindhoven University of Technology and is part of the research programme "oCPS" funded from the European Union's Horizon 2020 Framework Programme for Research and Innovation under grant agreement No. 674875.

A catalogue record is available from the Eindhoven University of Technology Library.
ISBN: 978-90-386-5277-1

Typeset using LaTeX

Reproduction: ADC Dereumaux

Cover photo: Miguel Á. Padriñán

Model-Based Supervisory Control Synthesis of Cyber-Physical Systems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op woensdag 26 mei 2021 om 13:30 uur

door

Aida Rashidinejad

geboren te Tehran, Iran

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. H. Nijmeijer
1e promotor:	dr.ir. M.A. Reniers
2e promotor:	prof.dr. M. Fabian (Chalmers University of Technology)
leden:	prof.dr. C.N. Hadjicostis (University of Cyprus)
	prof.dr. W.J. Fokkink
	prof.dr.ir. J.P.M. Voeten

Het onderzoek dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Dedicated to my parents.

Societal Summary

Cyber-physical systems (CPSs) exist everywhere in human life, from robotics and traffic control systems to smart grid and manufacturing systems. Generally speaking, any combination of physical processes, computer-based algorithms, and communication between those for control purposes makes a cyber-physical system. Many applications of CPSs are safety-critical such as autonomous vehicles or medical devices. For safety-critical CPSs, it should always be guaranteed that the system behaves within the specified behavior. Otherwise, the physical system could hurt itself, the people around it, or the environment. This thesis offers new methods to improve the supervisory control layer of a CPS, which is responsible to guarantee the safety of the system.

The communication in a CPS is through a network. Using a network brings many benefits by eliminating unnecessary wiring. However, it introduces challenges. Consider as an example a railroad crossing located in Eindhoven and controlled from Rotterdam via a network. To prevent accidents between trains and passing cars, a safety requirement is to guarantee that the gate is closed when a train arrives and is opened only after the train leaves. Using the supervisory control theory, a supervisor is achieved, guaranteeing that the system behaves within the safety requirement. Similar to other theoretical approaches, supervisory control theory assumes an ideal practical situation that does not necessarily exist in real life. For instance, the modeling framework only concerns the sequences of events, such as the arrival/departure of the train or closing/opening the gate, while the timing of the event occurrences is neglected. Moreover, the supervisory control theory assumes a control setup with complete synchronous interactions, i.e., the sensor information is immediately received by the supervisor, and the control commands are immediately received by the system. In contrast, when controlling systems over a network, time plays a significant role as communication delays are unavoidable. Neglecting the communication delays in theory may result in the failure of the obtained supervisor in implementation. In the railroad crossing example, the supervisor failure may lead to a catastrophic accident between a train and passing cars. The work presented in this thesis develops supervisory control theory for CPS by contributing to two directions; 1) developing networked supervisory control theory that deals with the effects of communication delays, and 2) developing the modeling framework for the supervisory control theory by incorporating real-time. This research is an important step towards more reliable supervisory control of CPS to guarantee that they behave in a safe manner.

Abstract

Cyber-physical systems (CPSs) integrate physical processes, computer-based algorithms, and communication between those for control purposes. A CPS has hybrid dynamics; a combination of both time-driven and event-driven dynamics. In this respect, a CPS usually involves two layers of control; low-level (continuous-variable) controllers that are mainly responsible to enhance the performance of the system in terms of exhibiting some desired behavior, and high-level (supervisory) controllers that are mainly responsible to guarantee the safety and liveness of the system. There are many applications of CPS for which the supervisory control layer is of significant importance. For instance, vehicle collision avoidance is a safety requirement for a platooning system that needs to be satisfied by the high-level controllers.

To guarantee the safety and liveness of a system, the concept of supervisory control theory (SCT) has been developed. SCT is based on the synchronous interactions between the plant and the supervisor. This assumption fails in practice when the supervisor, synthesized from theory, is implemented. More importantly, although network-based control in cyber-physical systems brings many advantages, it also involves challenges, as delays, packet reordering, packet losses, and the risk of cyber-attacks are introduced by and through network communication. The main objective of this thesis is to guarantee the safety and liveness of a CPS by developing model-based supervisory control in a setting where communication imperfections or the risk of cyber-attacks may appear.

As the first contribution, this thesis investigates supervisory control of discrete-event systems (DESs), represented by finite automata (FA), in an asynchronous setting. A supervisor, synthesized from conventional SCT, may fail in practice due to communication delays that may appear in the implementation but are neglected in theory. In this thesis, a supervisor is synthesized in an asynchronous supervisory control setting, where the interactions between the plant and the supervisor are not assumed to be synchronous. Besides delays, interleave-sensitivity and causality are other implementation problems that are considered by the proposed method.

Second, a networked supervisory control synthesis method is proposed in the modeling framework of timed DESs (TDESs). TDESs consider discrete-time information in DESs. The timing information is used to model the effects of communication delays. To synthesize a networked supervisor, a networked supervisory control framework is provided, where the supervisor interacts with the plant through communication channels that introduce delays. Besides delays, the problem of packet reordering is considered. Based on the proposed framework, a networked plant automaton is achieved, modeling the behavior of the plant under the effects of communication delays and packet reordering. The networked supervisor is then synthesized based on the networked plant.

Third, due to the scalability problem of discrete-time modeling, dense/real-time modeling is considered in DESs, where the plant is represented by a timed automaton (TA). A TA is an FA extended with a finite set of real-valued clocks. To model the timing behavior of TA, the accepting temporal conditions to switch between different modes, called locations, or stay in the current one are represented by clock constraints, indicated by guards and invariants, respectively. To deal with the infinite state space of a TA, the existing supervisory control approaches rely on abstracting a TA into an FA and apply the synthesis to the FA. Moreover, they achieve a supervisor that is only able to adjust the guards of a TA. In this thesis, a synthesis technique is proposed that abstracts a TA into an FA such that the event set of the FA includes the discrete events of the TA as well as an event representing a time delay. Time delays are considered to be preemptable by events from a given set of forcible events. By using the concept of forcible events, the supervisor is allowed to control a TA by not only adjusting the guards but also the invariants. In this way, the degree-of-freedom of the supervisor is increased, and so the resulting supervisor is less conservative compared to other techniques. For many applications, using abstractions results in finite state-spaces but still risks encountering the state-space explosion problem. To deal with this issue, this thesis also presents a supervisory control synthesis technique that is directly applicable to TA without any abstraction. Moreover, the concept of forcible events is used again to provide the supervisor with more control actions. To synthesize such a supervisor, an algorithm is proposed that iteratively strengthens the guards of controllable events and invariants of locations where time progress can be preempted by forcible events.

Fourth, a networked control system may also face the risk of cyber-attacks, which may cause catastrophic damage to the system. In this regard, this thesis also studies the effects of cyber-attacks in networked supervisory control of DES by providing a survey on existing methods and analyzing them to propose new research directions.

List of Publications

The results presented in this thesis are based on the following publications.

Peer-reviewed journal contributions

Aida Rashidinejad, Michel Reniers, and Martin Fabian, *Networked Supervisory Control Synthesis of Timed Discrete-Event Systems.* 2020, submitted.

Aida Rashidinejad, Michel Reniers, and Martin Fabian, *Supervisory Control Synthesis of Timed Automata Using Forcible Events.* 2020, submitted.

Peer-reviewed conference contributions

Aida Rashidinejad, Michel Reniers, and Lei Feng. *Supervisory Control of Timed Discrete-Event Systems Subject to Communication delays and Non-FIFO Observations.* Proceedings of 14th Workshop on Discrete Event systems, pp. 456-463, 2018, doi: 10.1016/j.ifacol.2018.06.340.

Aida Rashidinejad, Bart Wetzels, Michel Reniers, Liyong Lin, Yuting Zhu, and Rong Su. *Supervisory Control of Discrete-Event Systems under Attacks: an Overview and Outlook.* Proceedings of 18th European Control Conference, pp. 1732-1739, 2019, doi: 10.23919/ECC.2019.8795849.

Aida Rashidinejad, Michel Reniers, and Martin Fabian. *Supervisory Control of Discrete-Event Systems in an Asynchronous Setting.* Proceedings of IEEE 15th International Conference on Automation Science and Engineering, pp. 494-501, 2019, doi: 10.1109/COASE.2019.8843274.

Aida Rashidinejad, Patrick van der Graaf, Michel Reniers, and Martin Fabian. *Nonblocking Supervisory Control of Timed Automata using Forcible Events.* Proceedings of 15th International Workshop on Discrete Event Systems, 2020, to appear.

Aida Rashidinejad, Patrick van der Graaf, and Michel Reniers. *Nonblocking Supervisory Control Synthesis of Timed Automata using Abstractions and Forcible Events.* Proceedings of 16th International Conference on Control, Automation, Robotics, and Vision, pp. 1-8, 2020, doi:10.1109/ICARCV50220.2020.9305312.

Non peer-reviewed conference contributions

Aida Rashidinejad, Michel Reniers, and Maurice Heemels. *Networked Supervisory Control Synthesis of Discrete-Event Systems with Time-Delayed Non-FIFO Communication.* Proceedings of 36th Benelux Meeting on Systems and Control, pp. 50, 2017.

Aida Rashidinejad, and Michel Reniers. *Networked supervisory control synthesis of discrete-event systems with time-delayed non-FIFO communication.* ICT.OPEN 2017.

Aida Rashidinejad, and Michel Reniers. *Examples of networked supervisory control synthesis.* ICT.OPEN 2019.

Contents

Chapter 1

Introduction

In theory, theory and practice are the same. In practice, they are not.
–Albert Einstein

In this chapter, cyber-physical systems and the motivation of this research are introduced. Subsequently, the problem of supervisory control synthesis for cyber-physical systems is described. This chapter continues by providing the research questions and main contributions. Finally, the outline of this thesis is provided.

1.1 Motivation

A *cyber-physical system* (CPS) is the integration of physical processes, computer-based algorithms, and communication between those for control purposes. CPSs have widespread practical applications; from smart manufacturing and robotics to health care and medicine [Baheti and Gill 2011; Gunes et al. 2014; Reniers et al. 2017; Thoben et al. 2017]. Basically, any complex engineered system, integrating the embedded technologies (cyber part) into the physical world is a CPS [Gunes et al. 2014].

A system, in general, may have *time-driven* dynamics, *event-driven* dynamics, or a combination of both, which is referred to as a *hybrid system* [Cassandras and Lafortune 2009; Heemels et al. 2009]. Typically, a CPS has hybrid dynamics; physical processes are time-driven, represented by dynamical equations, and the cyber part or the control flow is event-driven, which can be represented by, for instance, a *finite-state machine* (FSM) [Derler et al. 2011].

As a hybrid system, a CPS involves different layers of control. Inspired from the definition of a CPS in Gunes et al. [2014] and control layers of a hybrid system in Cassandras and Lafortune [2009], Figure 1.1 provides a high-level view of the control layers of a CPS, described as follows:

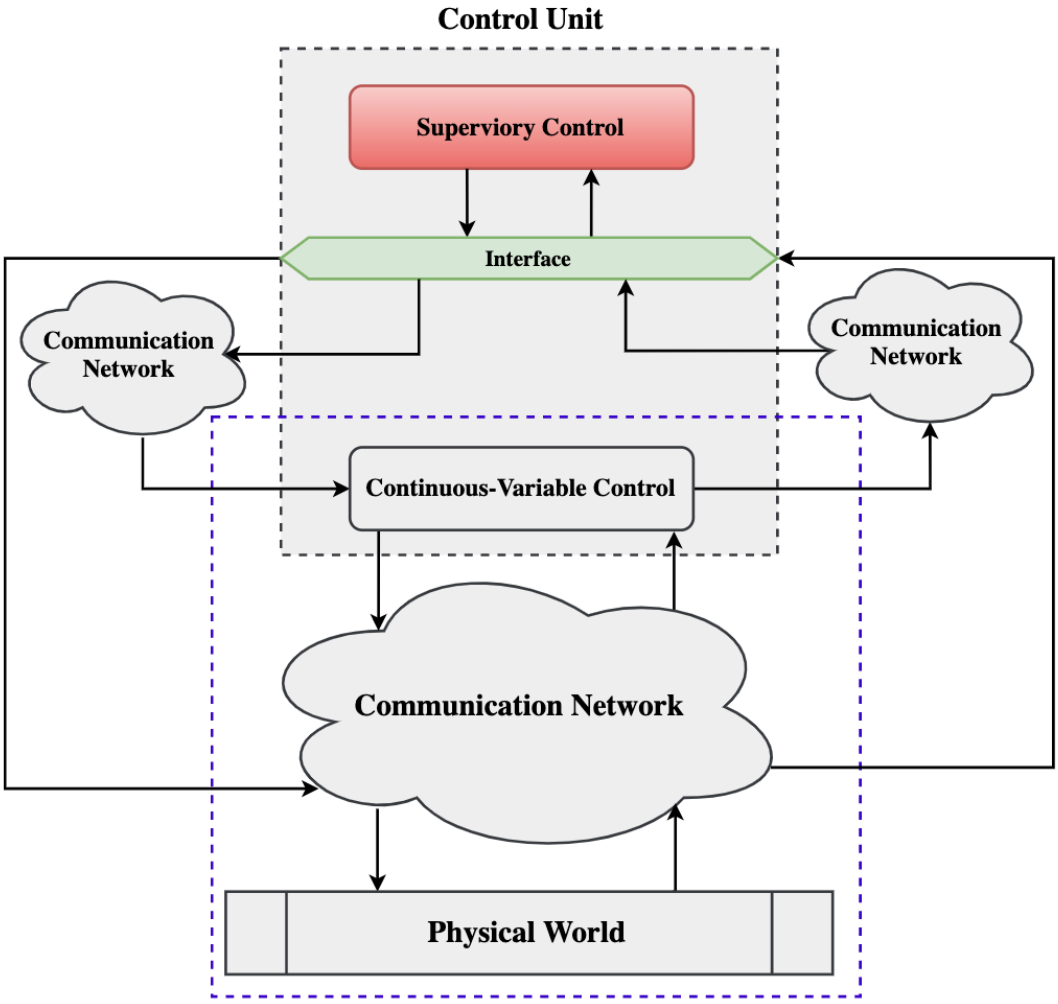


Figure 1.1: Control layers of a CPS. Note that there is a single communication network between the control unit and the physical world.

- *Continuous-variable control* (time-driven): low-level controllers, which are responsible for low-level control requirements mainly defined to improve the performance of the system in terms of achieving a desired behavior. Considering a multi-robot system as an example, following the desired trajectory would be a low-level control requirement.
- *Supervisory control* (event-driven): high-level controllers, referred to as supervisors, which are responsible for high-level control requirements that mainly concern the safety and liveness of the system. In the example of a multi-robot system, avoiding the collision between the robots and reaching some target location, may be given as safety and liveness requirements, respectively.

As depicted in Figure 1.1, the supervisory control layer communicates with the lower level through an *interface*. The output signals from the sensors and the continuous-variable controllers are abstracted to events through the interface. The commands, sent by the supervisors are also in the form of events, which go through the interface to be translated as input signals to the actuators or the set-points for the continuous-variable controllers [Cassandras and Lafortune 2009].

Although generally a CPS has hybrid dynamics, it may be modeled as a system with only time-driven or event-driven dynamics. Each modeling framework, either time-driven or event-driven, highlights certain features of a CPS. The choice of the modeling framework depends on the system analysis purpose [Baheti and Gill 2011].

For some CPSs, it may be enough to solely improve the performance using continuous-variable controllers. However, there exist many applications of CPSs for which the safety of the system is of significant importance. Such systems are referred to as safety-critical applications of CPSs. Medical devices, aircraft, and automobiles are among examples of such systems [Alur 2015; Sha et al. 2008].

1.2 Problem Description

The focus of this thesis is on the supervisory control layer of a CPS. In this respect, the lower level of a CPS, indicated in blue in Figure 1.1, is considered to be given as the (uncontrolled) plant, which is viewed as a system with event-driven dynamics through the interface. A large amount of literature is available for abstracting a hybrid system into an event-driven setting [Alur et al. 2000; Koutsoukos et al. 2000; Tiwari 2008].

As discussed in Cassandras and Lafortune [2009], the abstraction could be untimed (logical), or incorporate timing information and is then referred to as a timed abstraction. An untimed abstraction results in a modeling framework called a *discrete-event system* (DES) in this thesis. A DES is represented by a finite automaton (FA), an automaton with a finite set of states, and a finite set of events, where the state transitions are based on the event occurrences. In DESs, only the ordering of events matters, and the timing of events is neglected. Timed abstraction results in a modeling framework called a *real-time DES* (RTDES) [Khoumsi and Nourelfath 2002]. RTDESs may incorporate discrete-time modeling in DESs, referred to as a *timed DES* (TDES) [Brandin and Wonham 1994], or dense-time modeling, referred to as a *timed automaton* (TA) [Alur and Dill 1994]. A TDES is a DES in which the passage of a unit of time is represented by an event. A TA is an FA extended with a finite set of real-valued clocks with time-driven dynamics. In a TA, state transitions are not only based on event occurrences but also based on clock values or the time that events occur.

Supervisory control theory (SCT) was initiated by Ramadge and Wonham [1987], which we may refer to as *conventional SCT*. Given the model of an (uncontrolled) plant and the desired behavior for that plant, a supervisor synthesis procedure is developed based on model-based techniques.

As depicted in Figure 1.2, *model-based synthesis* starts from the system requirements R. Next, through the document D, these requirements are decomposed for the plant R_P and the supervisor R_S, where R_S is referred to as control requirements. Based on R_P, the plant model P is designed through the document D_P. However, the control requirements R_S are formally modeled. SCT now uses the model of the plant P and the control requirements R_S to synthesize a supervisor. The objective is to restrict the behavior of the plant such that the supervised plant (plant together with the supervisor) fulfills the desired behavior (given as control requirements).

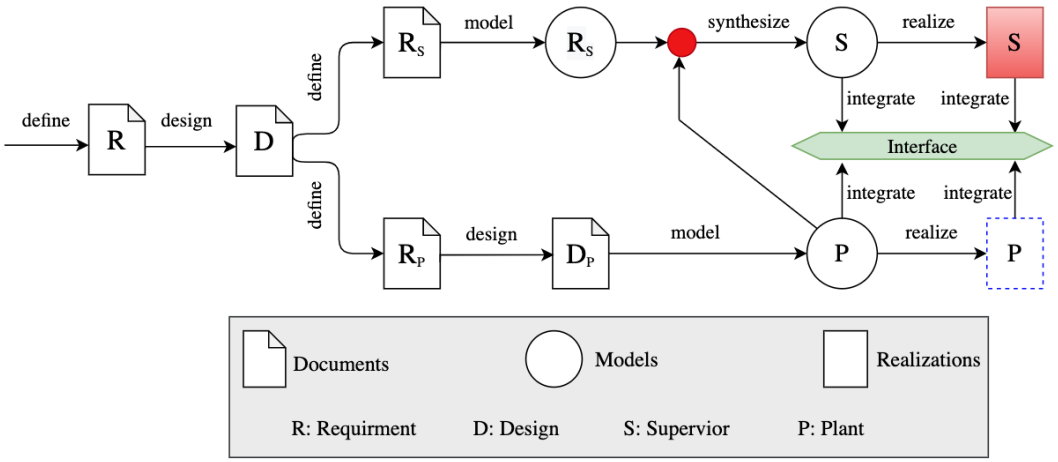


Figure 1.2: Model-based synthesis, adapted from [Baeten et al. 2016].

In the conventional supervisory control setting, the plant generates all events, while the supervisor can disable some of the events and observes synchronously the execution of events in the plant [Cassandras and Lafortune 2009; Ramadge and Wonham 1984]. In other words, conventional SCT assumes full synchronization between the supervisor and the plant. A more in-depth discussion on conventional SCT is given in Chapter 2 and Chapter 6 in the framework of DES, in Chapter 3 in the framework of TDES, and in Chapter 4 and Chapter 5 in the framework of TA.

Model-based supervisory control appeared in many industrial applications, including manufacturing systems [Balemi 1992; Hellgren et al. 2001], healthcare systems [Theunissen et al. 2013], automated service composition [Atampore et al. 2016], waterway lock systems [Reijnen et al. 2020], baggage handling systems [Swartjes et al. 2017], and lithography systems [van Putten et al. 2020]. In most of these applications, the supervisor is implemented in a *programmable logic controller* (PLC).

Supervisory control of CPSs is an ongoing research topic among researchers from both academia and industry as it involves challenges in both theory and practice, mainly introduced by network-based communication. This thesis aims to address these challenges and provide solutions to them by using model-based supervisory control synthesis techniques.

1.3 Research Questions

This thesis investigates model-based supervisory control synthesis of CPSs. Considering Figure 1.1, the questions that may arise in this respect could be mostly related to the supervisory control layer (Research Question 1), the communication network (Research Question 2 and Research Question 3), and the level of model abstraction through the interface (Research Question 2). In the following, the questions to which this thesis tries to find answers are discussed in more detail.

> **Research Question 1:** *How can the supervisory control theory be improved in such a way that the synthesized supervisor copes with the problems that may appear in the PLC-implementation?*

As previously mentioned, the conventional SCT is developed based on the assumption that the plant and the supervisor communicate synchronously. Moreover, it is assumed that the plant generates all the events, and the supervisor may only disable some of them [Fabian and Hellgren 1998]. As reported in the literature of SCT, a supervisor, synthesized from the theory may face several problems when implemented in practice, specifically in a PLC, mainly because these assumptions fail [Balemi 1992; Basile and Chiacchio 2007; Fabian and Hellgren 1998; Leal et al. 2009; Prenzel and Provost 2018; Zaytoon and Riera 2017]. As a result, the implemented supervisor may fail to satisfy the control requirements as it is promised to do in theory. The question here is how to synthesize a supervisor that will not fail in implementation, where these problems may occur. By providing an answer to Research Question 1, this research aims to close the gap between the supervisor, obtained from theory with its implementation in real-life.

> **Research Question 2:** *How can a supervisor be synthesized that is able to control a system over a communication network?*

As depicted in Figure 1.1, the communication between the physical world and the control unit in a CPS is via a network. The internet of things (IOT) brings many advantages. However, it also introduces challenges. In a networked control system, time delays are unavoidable, and they have a high impact on the performance of the control system [Heemels et al. 2010]. So far, extensive research has been performed on networked control for systems with time-driven dynamics [Antsaklis and Baillieul 2007; Gupta and Chow 2010; Heemels et al. 2010]. However, networked supervisory control is a relatively new emerging topic in the literature [Balemi 1992; Lin 2014, 2020; Park and Cho 2006; Xu et al. 2017; Zhu et al. 2019b]. To handle the effects of communication delays, it is first required to model them. In this regard, the main question is how to incorporate timing information in the networked supervisory control synthesis of an event-driven system. While the main issue is handling the effects of communication delays, there might be other problems introduced by the communication network like packet reordering [1] or packet losses. Here, the question is whether these problems can also be taken care of in networked

[1] The effect of packet reordering is studied as the interleave sensitivity problem in Chapter 2 and non-FIFO communication in Chapter 3.

supervisory control synthesis approaches.

> **Research Question 3:** *How can networked supervisory control be affected by cyber-attacks?*

Besides communication delays, dealing with the effects of cyber-attacks is of significant importance in the area of networked control. Developing attack-prevention techniques in the area of supervisory control has recently gained considerable attention [Carvalho et al. 2018; Lima et al. 2018; Meira-Góes et al. 2020; Su 2018; Wakaiki et al. 2017]. The question here is whether the effects of attacks can be similar to the impact of communication delays. If so, how the methods proposed to handle the communication delays and cyber-attacks can be inspired by each other.

To clarify how these research questions are answered throughout the thesis, we put them in a framework consisting of the following dimensions:

- Supervisory control robustness: questioning how the conventional SCT can be improved in order to deal with the problems that the communication network may introduce in a CPS.
- Modeling framework: questioning whether the level of abstraction, provided by the interface, preserves enough information to take care of the problems that the communication network may introduce in a CPS.

1.4 Main Contributions

As a CPS is hybrid, ideally, a networked supervisory control approach is required for a CPS, modeled as a *hybrid automaton* (HA); a modeling framework that involves both event-driven and time-driven dynamics. Moreover, the main issues that should be taken care of by the networked supervisor are communication delays and cyber-attacks; packet reordering and packet losses are other minor problems. This thesis makes the first steps to further develop to the ideal situation by the following contributions.

Contribution 1 (From Implementation to Theory): While providing supervisory control synthesis techniques, one should be aware of the issues that may cause the synthesized supervisor to fail in practice. In conventional SCT, the plant communicates with the supervisor synchronously. The main issue that arises in practice is related to the *inexact synchronization* introduced by the PLC-implementation of the supervisor [Fabian and Hellgren 1998]. In this thesis, an asynchronous supervisory control theory is proposed in the framework of DES. Besides inexact synchronization, the problems of *interleave sensitivity* and *causality* are taken care of by the proposed technique. This contribution relates to Research Question 1.

Contribution 2 (Delays in Networked Supervisory Control): As previously mentioned, a CPS is a hybrid system in general, which is abstracted to an event-driven system for

supervisory control purposes. Existing networked supervisory control approaches usually work with DES [Balemi 1992; Lin 2014; Park and Cho 2006; Xu et al. 2017; Zhu et al. 2019b]. However, this level of abstraction may not be enough to consider the effects of communication delays as they need to be modeled based on time. To incorporate discrete-time modeling in a DES, the concept of TDESs has been introduced in Brandin and Wonham [1994]. For TDESs, the conventional supervisory control theory has already been proposed in Brandin and Wonham [1994], in which again the plant communicates with the supervisor synchronously. In this thesis, a networked supervisory control theory is proposed in the framework of TDES. The timing information is used to model the effects of communication delays. Besides communication delays, the problem of packet reordering is taken care of by the proposed technique. This contribution relates to Research Question 2.

Contribution 3 (Real-Time Supervisory Control): To make the model of time more realistic and make a step closer to the goal, dense-time information is incorporated in DESs, given as TA. Supervisory control of TA has already been studied in Alur and Dill [1994], Asarin et al. [1998], Maler et al. [1995], Tripakis and Altisen [1999], and Wong-Toi and Hoffmann [1991]. However, it still needs to be improved to be suited for CPS applications with respect to the following issues:

- Conservative Supervisor: the available techniques usually result in a supervisor that restricts the behavior of the plant much more than needed.
- State-space explosion: as a TA may have an infinite state space, the available methods use abstraction to obtain a finite state space. Using this abstraction often results in a very large state space.

In this thesis, two supervisory control approaches are proposed for TA. First, we focus on solving only the conservativeness problem. Being inspired by what we achieved in the first step, we come up with the second approach, which solves both the conservativeness and state-space explosion problems at the same time. These contributions relate to Research Question 2.

Contribution 4 (Attacks in Networked Supervisory Control): Besides communication delays, cyber-attacks may occur in controlling systems over a network, which could cause catastrophic damage to the system [Pasqualetti et al. 2013; Teixeira et al. 2012]. Depending on attack types, cyber-attacks could have similar effects as communication delays or packet losses. This thesis provides an overview of the existing approaches, dealing with the effects of cyber-attacks in the context of supervisory control of DES [Carvalho et al. 2018; Góes et al. 2017; Lima et al. 2017; Meira-Góes et al. 2020; Su 2018; Wakaiki et al. 2017; Yao et al. 2020]. It provides a framework to categorize these approaches. Based on the proposed framework, it identifies missing pieces, possible links to networked supervisory control methods that deal with delays or losses, and also determine new directions for further research to improve the existing literature on networked supervisory control. This contribution relates to Research Question 3.

These main contributions complete the framework introduced in Section 1.3 by adding the following sub-categories, ordered in terms of complexity:

- Supervisory control robustness: based on the level of robustness of the supervisor, supervisory control of CPS is divided into the following sub-categories:
  1. Conventional supervisory control: plant and supervisor communicate in a synchronous manner.
  2. Supervisory control under delays: plant and supervisor communicate over a network, introducing delays.
  3. Supervisory control under attacks: plant and supervisor communicate over a network, introducing the risk of cyber-attacks.
- Modeling framework: based on the level of abstraction of a CPS provided by the interface, supervisory control of a CPS may be performed in the following frameworks:
  1. DESs: systems with a finite set of discrete states and a finite set of discrete events in which the state transitions depend entirely on event occurrences.
  2. TDESs: DESs that incorporate discrete-time information of event occurrences, so that the tick of the clock is also considered as a discrete event.
  3. TA: DESs that incorporate dense-time information of event occurrences, so that states include timing information, and state transitions depend not only on event occurrences but also on the time that events occur.
  4. HA: DES that incorporate time-driven dynamics to the evolution of events, so that states include dynamical equations and state transitions depend not only on event occurrences but also on the system time-driven dynamics.

1.5 Outline of the Thesis

The structure of this thesis is as follows. Chapter 2 investigates asynchronous supervisory control synthesis. In this chapter, the PLC-implementation issues, namely inexact synchronization, interleave sensitivity, and causality are taken care of by the synthesized supervisor in the framework of discrete-event systems. In Chapter 3, the problem of networked supervisory control synthesis is studied. To deal with the effects of communication delays, a networked supervisory control synthesis approach is presented in the framework of timed discrete-event systems. Besides communication delays, the networked supervisor deals with the packet reordering problem. Chapter 4 and Chapter 5 focus on supervisory control synthesis in the framework of timed automata (TA). Chapter 4 provides a synthesis approach on the basis of abstracting time. Although the synthesis is provided on the abstracted level, the synthesized supervisor is translated back into a TA by introducing a time-refinement method. As abstraction may result in a large state-space for many practical applications, the method proposed in Chapter 4 is further improved in Chapter 5. There, a synthesis method is proposed for timed automata that does not need any abstraction. Chapter 6 provides an overview of the supervisory control synthesis approaches, in the framework of discrete-event systems, aiming to take care of the

effects of cyber-attacks. This chapter also investigates possible links to other supervisory control methods that deal with problems with similar effects to cyber-attacks and provides new research directions. Finally, Chapter 7 answers the research questions and gives recommendations for future research.

An overview of the thesis is depicted in Figure 1.3. It categorizes the existing methods, contributions provided in the thesis, and research that still needs to be conducted based on the dimensions and subcategories introduced in Section 1.3 and Section 1.4, respectively.

Chapter 2 is based on Rashidinejad et al. [2019a], Chapter 3 is based on Rashidinejad et al. [2021a], Chapter 4 is based on Rashidinejad et al. [2020a], Chapter 5 is based on Rashidinejad et al. [2021b], and Chapter 6 is based on Rashidinejad et al. [2019b], where extra examples have been used to enhance understanding of the approaches. Moreover, to enhance readability, all technical lemmas and proofs are given in the appendices. As depicted in Figure 1.3, each chapter is related to a specific modeling framework and supervisory control robustness level. Each chapter provides its introduction to the existing methods and the relevant preliminaries. Consequently, each chapter can independently be studied by readers of this thesis.

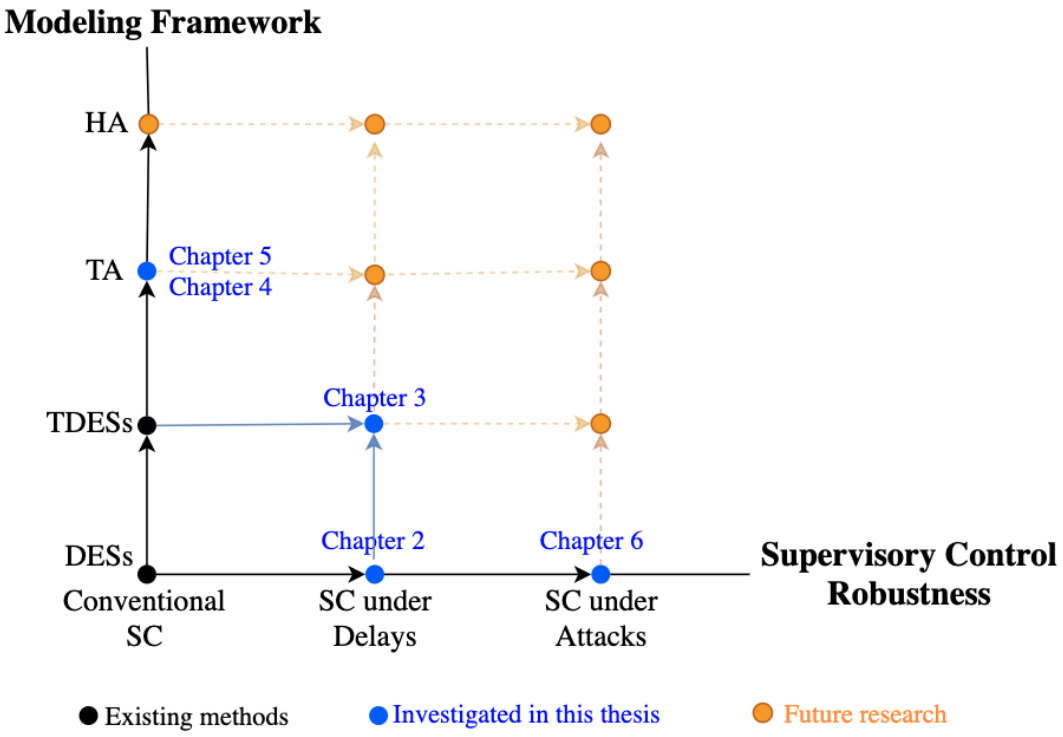


Figure 1.3: Overview of thesis in the proposed framework.

Chapter 2

Asynchronous Supervisory Control of Discrete-Event Systems

In conventional supervisory control theory, a plant and supervisor are supposed to work synchronously such that enabling an event by the supervisor, execution of the event in the plant, and observation of the executed event by the supervisor all occur at once. Therefore, these occurrences are all captured by means of a single event. However, when a supervisor synthesized from conventional supervisory control theory is implemented in real life, it will face problems since exact synchronization can hardly happen in practice due to delayed communications. In this chapter, we propose a synthesis technique to achieve a supervisor that does not face the problems caused by inexact synchronization. For this purpose, we first introduce an asynchronous setting in which enablement, execution, and observation of an event do not occur simultaneously but with some delay. We present a model representing the behavior of the plant in the asynchronous setting, which we call the asynchronous plant. For the asynchronous plant, we present an algorithm synthesizing an asynchronous supervisor that satisfies (asynchronous) controllability and nonblockingness.

2.1 Introduction

Discrete-event systems (DESs) are systems with a discrete set of states in which transitions take place in association with instantaneous events. Many types of physical systems can be modeled as DESs, including manufacturing systems, traffic systems, and communication networks [Cassandras and Lafortune 2009; Wonham 2015]. Such systems are typically

This chapter is based on Rashidinejad et al. [2019a]

supervised in order to satisfy some control requirements. Supervisory control theory (SCT) has been developed to automatically synthesize such a supervisor from models of the uncontrolled system and the control requirements [Cassandras and Lafortune 2009; Wonham 2015]. In a supervisory control setting, control commands are sent from the supervisor to the actuators in the plant through the control channel, and events occurring in the plant are observed by the sensors and sent to the supervisor through the observation channel. Several problems that arise in implementation of supervisors have been reported in the literature as briefly discussed in the following [Basile and Chiacchio 2007; Fabian and Hellgren 1998; Leal et al. 2009; Prenzel and Provost 2018; Zaytoon and Riera 2017]:

- Avalanche effect: this is the problem of multiple state transitions occurring on the same event leading to instantaneously passing through intermediate states so that actions from those states are not executed.
- Interleave sensitivity: this problem occurs when different interleavings of events can be executed at a state, and the supervisor needs to make decisions based on the order of events that it observes. The problem arises when events may not necessarily be observed in the same order as they have been executed in the plant.
- Causality: SCT assumes all events can occur spontaneously in the plant, and the supervisor may only disable a subset of them. However, in practice, events that are controllable need to be commanded by the supervisor, otherwise they cannot occur.
- Choice: SCT achieves a supervisor that is maximally permissive, meaning that it gives the plant the greatest amount of freedom within the control requirements. Therefore, in many cases, the supervisor should make a choice between alternative transitions that are possible at a state, and this will be highly dependent on the implementation.
- Inexact synchronization: in SCT, a supervisor is synthesized based on the assumption of synchronous interactions between the plant and supervisor. In other words, a supervisor is assumed to immediately observe an event as it is executed by the plant, and the plant receives a control command immediately after it has been sent from a supervisor. However, such a supervisor will be implemented in an asynchronous setting, where sending and receiving data are subject to delays. Hence, the synchronous assumption does not typically hold.

Here, we focus on the problem caused by inexact synchronization. Additionally, we take into account the problems of causality and interleave sensitivity by allowing the plant to execute a controllable event only if it is enabled (commanded) by a supervisor, and also allowing consecutive events to be observed in any possible order.

The problem caused by inexact synchronization was first addressed in Balemi [1992]. It arises in the situation where the occurrence of an uncontrollable event invalidates a control command (or the selection of a controllable event). This problem is caused by a communication delay where the occurrence of an uncontrollable event is interpreted as a communication delay unit. Consider a state of the plant where both a controllable event and an uncontrollable event are enabled. If there is a communication delay, the supervisor might send the controllable event while the plant transits to another state on

the uncontrollable event. Then, the control command may arrive when the plant is in a state that invalidates that command.

To solve the problem caused by communication delays, Balemi introduced the notion of *delay insensitive language*. A language is delay insensitive if a control command is not invalidated by an uncontrollable event. In other words, any control command sent by the supervisor should be acceptable by the plant both before and after the uncontrollable event. If a supervised plant is delay insensitive, then the achieved supervisor does not face any problem caused by inexact synchronization. However, this condition is not met by most applications.

The condition of delay insensitivity was also used in Malik [2002] called $\Sigma_u - \Sigma_c$-*commuting* condition. This condition has been further generalized in Basile and Chiacchio [2007] and Park and Cho [2006] for a sequence of uncontrollable events, representing a bounded observation delay.

Besides the inexact synchronization, the causality problem was also considered by Balemi as he uses an *input/output semantics* for the plant [Balemi 1992]. In this input/output perspective, controllable events are considered as inputs to the plant and uncontrollable events are the outputs or responses generated by the plant.

In Fabian and Hellgren [1998], a condition for *interleave insensitivity* was introduced. In this case, having an implementable supervisor is limited to applications that require the same control command after any interleaving of a sequence of uncontrollable events. In Basile and Chiacchio [2007], delay insensitivity and interleave insensitivity are captured in a single definition as *delay interleave insensitivity*.

If we assume that a control command is enabled until a disablement command is received from a supervisor, then delays in control and observation channels may have different effects and need to be investigated separately [Lin 2014; Xu and Kumar 2008]. To consider the effect of delays, in Xu and Kumar [2008], a condition called *bounded-delay implementability* has been introduced. This condition takes into account the effects of observation and control delays on a requirement behavior by achieving a delayed version of the requirement. In the case that the delayed version of the requirement stays within the requirement (without delays), it can be satisfied in an asynchronous setting as well. Furthermore, in Lin [2014], new observability and controllability conditions under delays are introduced as *delay observability* and *delay controllability*, respectively. A supervisor can be synthesized for a requirement that satisfies these conditions. However, the drawback of imposing conditions on the requirement is that these conditions disqualify many applications.

The method presented in this chapter does not consider any restrictive condition to be satisfied by the plant or the requirement. The objective is to present a method to synthesize a supervisor taking into account the following situations that may exist in practice:

1. A controllable event can be executed in the plant only if it is commanded (enabled) by the supervisor. Also, any event executed in the plant is observable to the supervisor.
2. An uncontrollable event is not commanded (enabled) by a supervisor, and it only occurs spontaneously in the plant.

3. A control command sent by the supervisor may not necessarily be accepted by the plant, and in this case the command will stay in the control channel since in the asynchronous setting there is no deadline for an enabling event to reach the plant.
4. The observation of an event, controllable as well as uncontrollable, may occur immediately after being executed in the plant or at some point in the future.
5. Consecutive events that occur in the plant may be observed by the supervisor in any possible order.

The rest of the chapter is organized as follows. The conventional supervisory control approach is summarized in Section 2.2. In Section 2.3, we present the asynchronous supervisory control setting, and we introduce an asynchronous composition operator to achieve an asynchronously supervised plant in this setting. Afterwards, in Section 2.4, we present a method for transforming a plant into an asynchronous plant modeling the behavior of the plant as asynchronously observed and controlled by the supervisor. Based on the asynchronous plant, an algorithm is proposed to synthesize an asynchronous supervisor that guarantees (asynchronous) controllability and nonblockingness. Finally, Section 2.5 concludes this chapter.

2.2 Background

A DES G is an automaton that is formally represented as a quintuple

$$G = (A, \Sigma, \delta, a_0, A_m), \tag{2.1}$$

where A, Σ, $\delta : A \times \Sigma \to A$, $a_0 \in A$, and $A_m \subseteq A$ stand for the set of states, the set of events, the (partial) transition function, the initial state, and the set of marked states, respectively. An automaton with a finite set of states and a finite set of events is called a finite automaton [Hopcroft et al. 2006]. The notation $\delta(a, \sigma)!$ denotes that δ is defined for state a and event σ, i.e., there is a transition from state a with label σ to some state. The transition function is generalized to words in the usual way: $\delta(a, w) = a'$ means that there is a (possibly empty) sequence of subsequent transitions from state a to the state a' that together make up the word $w \in \Sigma^*$. Note that for the empty sequence of subsequent transitions $w = \varepsilon$: $\delta(a, w) = a$. Starting from the initial state, the set of all possible words that may occur in G is called the language of G and is denoted by $L(G) := \{w \in \Sigma^* \mid \delta(a_0, w)!\}$. Furthermore, for a state $a \in A$, the function $Reach(a)$ gives the set of states reachable from state a; $Reach(a) := \{a' \mid \exists w \in \Sigma^*, \delta(a, w) = a'\}$. States from which it is possible to reach a marked state are said to be nonblocking. An automaton is nonblocking when each state reachable from the initial state is nonblocking; for each $a \in Reach(a_0)$, $Reach(a) \cap A_m \neq \varnothing$. Moreover, in this chapter, we frequently use the natural projection operator [Cassandras and Lafortune 2009].

Definition 2.1 (Natural projection)**.** For sets of events Σ and $\Sigma' \subseteq \Sigma$, $P_{\Sigma'} : \Sigma^* \to \Sigma'^*$ is defined as follows: for $e \in \Sigma$ and $w \in \Sigma^*$

$$P_{\Sigma'}(\epsilon) := \epsilon,$$
$$P_{\Sigma'}(we) := \begin{cases} P_{\Sigma'}(w)e & \text{if } e \in \Sigma', \\ P_{\Sigma'}(w) & \text{if } e \in \Sigma \setminus \Sigma'. \end{cases}$$

The definition of the natural projection can be extended to languages; given a language $L \subseteq \Sigma^*$, $P_{\Sigma'} : \Sigma^* \to \Sigma'^*$ maps it to a set of words from Σ'^* where $\Sigma' \subseteq \Sigma$ such that $P_{\Sigma'}(L) := \{w' \in \Sigma'^* \mid \exists w \in L, P_{\Sigma'}(w) = w'\}$ [Cassandras and Lafortune 2009]. ■

Note that, although natural projection is an operation that is generally defined for languages, it is also possible to apply it on automata [Ware and Malik 2008]. For an automaton with event set Σ, $P_{\Sigma'}$ first replaces all events not from Σ' by ε. Then, the achieved automaton becomes deterministic again, using a determinization algorithm such as the one introduced in Hopcroft et al. [2006].

From now on, we assume that the plant is given as a deterministic finite automaton G in which all events are observable. However, a subset of events may be uncontrollable, denoted by $\Sigma_{uc} \subseteq \Sigma$. The set of controllable events is then given by $\Sigma_c = \Sigma \setminus \Sigma_{uc}$. To satisfy nonblockingness, a supervisor is required to be synthesized for the plant G. A conventional supervisor S is also a DES with the same event set as G. The conventional supervisory control setting is depicted in Figure 2.1.

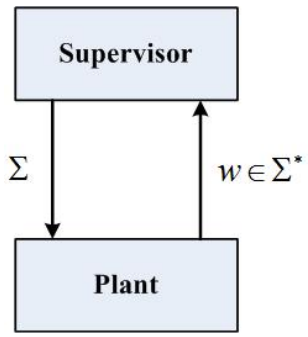


Figure 2.1: Conventional supervisory control (synchronous setting).

Here we focus on solving the basic synthesis problem to achieve a supervisor satisfying controllability and nonblockingness. As described in Flordal et al. [2007], control requirements can be translated to the plant. By applying the basic synthesis, a supervisor is achieved that fulfills the requirements, and it satisfies controllability and nonblockingness.

In conventional SCT, the plant and supervisor are supposed to work synchronously, and the automaton modeling the supervised plant is obtained by applying the synchronous composition operator denoted by $S||G$. Generally, in the synchronous composition of two automata, a shared event can be executed only when it is enabled by both automata, and a non-shared event can be executed if it is enabled by one of the automata. Since S is assumed to have the same event set as G, each event will be executed in $S||G$ only if the supervisor allows it. Since uncontrollable events enabled in G can never be disabled by

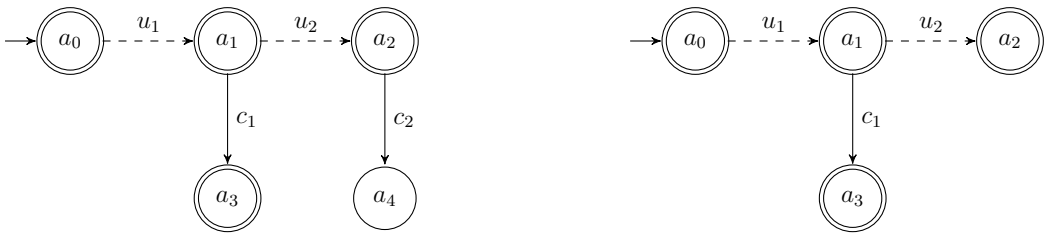
S, they should always be allowed in $S||G$, and this can be checked by the controllability condition given in Definition 2.2.

Definition 2.2 (Conventional Controllability [Wonham 2015])**.** For any DES G controlled by a conventional supervisor S, $S||G$ is controllable w.r.t. G if for any $w \in L(S||G)$ and $u \in \Sigma_{uc}$, whenever $wu \in L(G)$ then $wu \in L(S||G)$. ■

Note that the statement $S||G$ is controllable w.r.t. G is equivalent to S itself being controllable w.r.t. G.

In the following example, we explain the problem caused by inexact synchronization.

Example 2.1 (Motivating example from [Fabian and Hellgren 1998])**.** Consider the plant G, given in Figure 2.2a, for which u_1 and u_2 are uncontrollable events (indicated by dashed lines), and c_1 and c_2 are controllable events (indicated by solid lines). The state a_4 is blocking (marked states are indicated by double circles) and needs to be avoided by a supervisor. Using conventional supervisory control synthesis, the supervisor depicted in Figure 2.2b is obtained. As described in Fabian and Hellgren [1998], the problem appears when after the execution of u_1, S enables c_1; however, u_2 occurs in G, which invalidates the control command sent by S (because G cannot accept c_1 after executing u_2). Clearly, this problem does not occur if c_1 could occur after u_2 as well. This is actually the *delay insensitive language* condition introduced in Balemi [1992].



(a) Plant (b) Supervisor

Figure 2.2: The plant and the conventional supervisor from [Fabian and Hellgren 1998].

In order to synthesize a supervisor dealing with the problems that may arise in practice, we need to take into account the five conditions mentioned in Section 2.1 while synthesizing a supervisor.

2.3 Asynchronous Supervisory Control Setting

In the synchronous setting, enabling, execution, and observation of events are assumed to occur simultaneously, and so all of these are indicated by one and the same event.

As depicted in Figure 2.3, to indicate that the enabling, execution, and observation of events do not occur simultaneously in an asynchronous setting, for each event executed in the plant we introduce unique events representing the related *enabling events* and *observed events*.

Definition 2.3 (Enabling and Observed Events)**.** For a plant $G = (A, \Sigma, \delta, a_0, A_m)$, to each controllable event $\sigma \in \Sigma_c$ an enabling event $\sigma_e \in \Sigma_e$ and to any event $\sigma \in \Sigma$ an

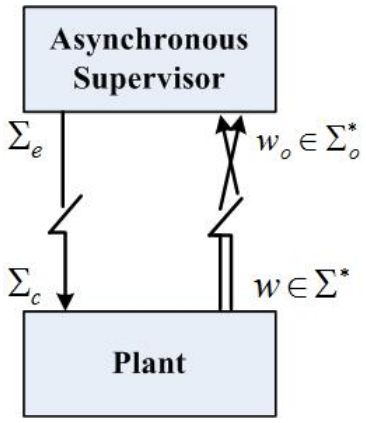


Figure 2.3: Asynchronous supervisory control setting.

observed event $\sigma_o \in \Sigma_o$ are associated. ■

As is clear from Figure 2.3, the plant and supervisor do not have the same event set in the proposed asynchronous setting. The event set of an asynchronous supervisor includes only the enabling and observed events. To achieve the supervised plant in the asynchronous setting, we need to define an asynchronous composition operator. To indicate how events may be observed in a supervisory control system, we assume that an event $\sigma \in \Sigma$ executed in the plant will be stored in the observation channel until being observed as $\sigma_o \in \Sigma_o$. Since it is assumed that events may not necessarily be observed in the same order as they have been executed in the plant, the observation channel is represented as a multiset. The size of the observation channel is considered to be limited to N_o. The multiset given in Definition 2.4 is a representation of the contents of the observation channel, which will be used in determining the occurrences of the observed events.

Definition 2.4 (Observation Channel Representation)**.** The set M is defined as $M = \{m \,|\, m : \Sigma \to \mathbb{N} \mid |m| \leq N_o\}$, where each element $m \in M$ is a multiset. Moreover, we define the following operations for all $m \in M$, and $\sigma, \sigma' \in \Sigma$:

- $m(\sigma)$ denotes the number of events σ in m.
- [] denotes the empty multiset, i.e, the function m with $m(\sigma) = 0$ for all $\sigma \in \Sigma$.
- $|m| = \sum_{\sigma \in \Sigma} m(\sigma)$ denotes the number of events in the observation channel represented by m.
- $m \uplus [\sigma]$ inserts σ to m if $|m| < N_o$ (the observation channel is not full). Formally, it denotes the function m' for which $m'(\sigma) = m(\sigma) + 1$ and $m'(\sigma') = m(\sigma')$ for $\sigma' \neq \sigma$. If $|m| = N_o$ (the observation channel is full), then the channel stays the same, i.e., $m' = m$.
- $m \setminus [\sigma]$ removes σ from m once. Formally, it denotes the function m' for which $m'(\sigma) = \max(m(\sigma) - 1, 0)$ and $m'(\sigma') = m(\sigma')$ for $\sigma' \neq \sigma$.
- $\sigma \in m$ denotes that σ is present in m, it holds if $m(\sigma) > 0$. ■

Similarly, enabling events sent by a supervisor are assumed to be stored in a control channel until being executed in the plant. However, since events will be executed based on

the order that they have been commanded, the control channel is represented by a list as given in Definition 2.5. Similar to the observation channel, the size of the control channel is considered to be limited to N_c.

Definition 2.5 (Control Channel Representation)**.** The set L is defined as $L = \{l \in \Sigma^*, |l| \leq N_c\}$. Moreover, we define the following operations for all $\sigma \in \Sigma$, and $l \in L$:

- $app(l, \sigma)$ adds the element σ to the end of l if $|l| < N_c$ (the channel is not full). For instance, for $l = \sigma_0\, \sigma_1\, \ldots\, \sigma_n$, $app(l, \sigma) = \sigma_0\, \sigma_1\, \ldots\, \sigma_n\, \sigma$. If $|l| = N_c$ (the control channel is full), then the channel stays the same.
- $head(l)$ gives the first element of l (for nonempty lists). For instance, for $l = \sigma_0\, \sigma_1\, \ldots\, \sigma_n$, $head(l) = \sigma_0$. $head(\varepsilon)$ is undefined.
- $tail(l)$ denotes the list after removal of its head (for nonempty lists). For instance, for $l = \sigma_0\, \sigma_1\, \ldots\, \sigma_n$, $tail(l) = \sigma_1\, \ldots\, \sigma_n$. $tail(\varepsilon)$ is undefined. ■

Based on the representations of observation and control channels, the *asynchronous composition operator* is presented in Definition 2.6 to obtain the asynchronously supervised plant.

Note that this asynchronous composition operator is only meaningful if G and AS are in accordance with the asynchronous supervisory control setting presented in this section.

Definition 2.6 (Asynchronous Composition Operator)**.** Consider the plant $G = (A, \Sigma, \delta, a_0, A_m)$ controlled by the supervisor $AS = (Y, \Sigma_{AS}, \delta_{AS}, y_0, Y_m)$ in an asynchronous setting. Then, the asynchronous composition of G and AS, denoted $AS||G$, gives the asynchronously supervised plant as the following automaton

$$AS||G = (Z, \Sigma_{ASP}, \delta_{ASP}, z_0, Z_m),$$

where

$$\begin{aligned} Z &= A \times Y \times M \times L, \\ \Sigma_{ASP} &= \Sigma_{AS} \cup \Sigma, \\ z_0 &= (a_0, y_0, [], \varepsilon), \\ Z_m &= A_m \times Y_m \times M \times L. \end{aligned}$$

ASP stands for asynchronously supervised plant. Moreover, for $a \in A$, $y \in Y$, $m \in M$, and $l \in L$, $\delta_{ASP} : Z \times \Sigma_{ASP} \to Z$ is defined as follows:

1. When an enabling event $\sigma_e \in \Sigma_e$ is executed in AS, it will be stored in l until being received by G. If $\delta_{AS}(y, \sigma_e)!$:

$$\delta_{ASP}((a, y, m, l), \sigma_e) = (a, \delta_{AS}(y, \sigma_e), m, app(l, \sigma)).$$

2. A controllable event $\sigma \in \Sigma_c$ can be executed in $AS||G$ if the enabling event σ_e was sent by the supervisor, and it is the first enabling event in the control channel to be executed. In addition, the event will be put in the observation channel. If $\delta(a, \sigma)!$ and $head(l) = \sigma$:

$$\delta_{ASP}((a, y, m, l), \sigma) = (\delta(a, \sigma), y, m \uplus [\sigma], tail(l)).$$

3. An uncontrollable event $\sigma \in \Sigma_{uc}$ can be executed in $AS||G$ only if it is executed in G. In addition, the event will be put in the observation channel to be received by AS later. If $\delta(a, \sigma)!$:

$$\delta_{ASP}((a, y, m, l), \sigma) = (\delta(a, \sigma), y, m \uplus [\sigma], l).$$

4. An observed event $\sigma_o \in \Sigma_o$ can be executed in $AS||G$ if $\sigma \in m$, subsequently σ is removed from m. If $\delta_{AS}(y, \sigma_o)!$ and $\sigma \in m$:

$$\delta_{ASP}((a, y, m, l), \sigma_o) = (a, \delta_{AS}(y, \sigma_o), m \setminus [\sigma], l). \quad \blacksquare$$

Note that the definition of nonblockingness and controllability stays the same as given in Section 2.2. However, because of introducing new sets of enabling and observed events, the formal definition of conventional controllability needs to be adapted for the asynchronous setting given as *asynchronous controllability* in Definition 2.7. A supervisor is (asynchronously) controllable for the plant if any uncontrollable event that could occur in the plant can be executed in the asynchronously supervised plant.

Definition 2.7 (Asynchronous Controllability)**.** Consider the plant G with event set Σ. Then, supervisor AS is asynchronously controllable for G if for all $w \in L(AS||G)$ and $u \in \Sigma_{uc}$, if $P_\Sigma(w)u \in L(G)$ then $wu \in L(AS||G)$. $\blacksquare$

In the proposed asynchronous setting, by definition, an uncontrollable plant event can occur whenever it is enabled in the plant, even though the plant is controlled by a supervisor (see Definition 2.6). Therefore, as we prove in Property 2.1, the asynchronous controllability condition is always guaranteed by the definition of the asynchronous composition operator. Note that, although the Σ_o events are uncontrollable, they are not enabled in the plant and therefore not considered as such in asynchronous controllability.

Property 2.1 (Controllable Asynchronous Supervisor)**.** For any AS and G, AS is asynchronously controllable for G.

Proof. See Appendix A.2.1. $\blacksquare$

Problem Statement: Given plant G, provide an asynchronous supervisor AS such that $AS||G$ is nonblocking.

2.4 Synthesis

In a real implementation, a supervisor determines which enabling events have to be sent based on what it has observed. Moreover, it could be possible that although the supervisor has sent an enabling event, the plant executes an uncontrollable event (or simply ignores the command because it cannot execute it). The enabling event will then stay in the control channel and block other enabling events waiting in the control channel to be executed in the plant. To achieve an asynchronous supervisor providing controllability and nonblockingness in the asynchronous setting, we do the synthesis on the *asynchronous plant*, indicated in Figure 2.4.

The asynchronous plant is a model for how events are executed in the plant based on enabling events, and also how observations of the executed events may occur in the asynchronous setting. To achieve the asynchronous plant automaton, we need to determine all possible cases that the enabling events could have been sent in the asynchronous setting. Since enabling events are based on observations, we first start by presenting a model for how the plant is actually observed in an asynchronous setting, which we call the *observed plant*, see Figure 2.4. The formal definition of the observed plant is presented in Definition 2.8.

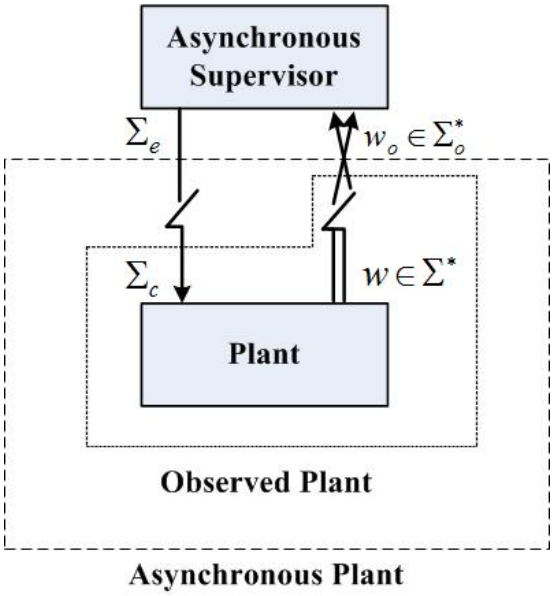


Figure 2.4: Asynchronous plant and observed plant.

Definition 2.8 (Observed Plant)**.** For a plant $G = (A, \Sigma, \delta, a_0, A_m)$, we define

$$\Upsilon(G, N_o) := (Q, \Sigma_{OP}, \delta_{OP}, q_0, Q_m),$$

where

$$Q = A \times M, \quad \Sigma_{OP} = \Sigma \cup \Sigma_o,$$
$$q_0 = (a_o, []), \qquad Q_m = A_m \times M.$$

OP stands for observed plant. The states of $\Upsilon(G, N_o)$ depend on the current states of the plant and of the observation channel. Initially, no event has occurred yet, and thus the observation channel is empty. Whenever an event occurs in the plant, it will be stored in the observation channel until it is observed by the asynchronous supervisor.

For $a \in A$, $m \in M$ and $\sigma \in \Sigma$, the transition function $\delta_{OP} : Q \times \Sigma_{OP} \to Q$ is defined as follows:

1. If $\delta(a, \sigma)!$
$$\delta_{OP}((a, m), \sigma) = (\delta(a, \sigma), m \uplus [\sigma]).$$
2. If $\sigma \in m$
$$\delta_{OP}((a, m), \sigma_o) = (a, m \setminus [\sigma]).$$

Note that when there are multiple events in the medium, these can be observed in any possible order. ■

Example 2.2. Let us again consider the plant from Example 2.1. For $N_o = 4$, the observed plant obtained from Definition 2.8 is given in Figure 2.5. Note that each state is a pair of (a, m) with a referring to a state of the plant (given inside the circles in Figure 2.5) and m referring to the current status of the observation channel (given close to the circles in Figure 2.5).

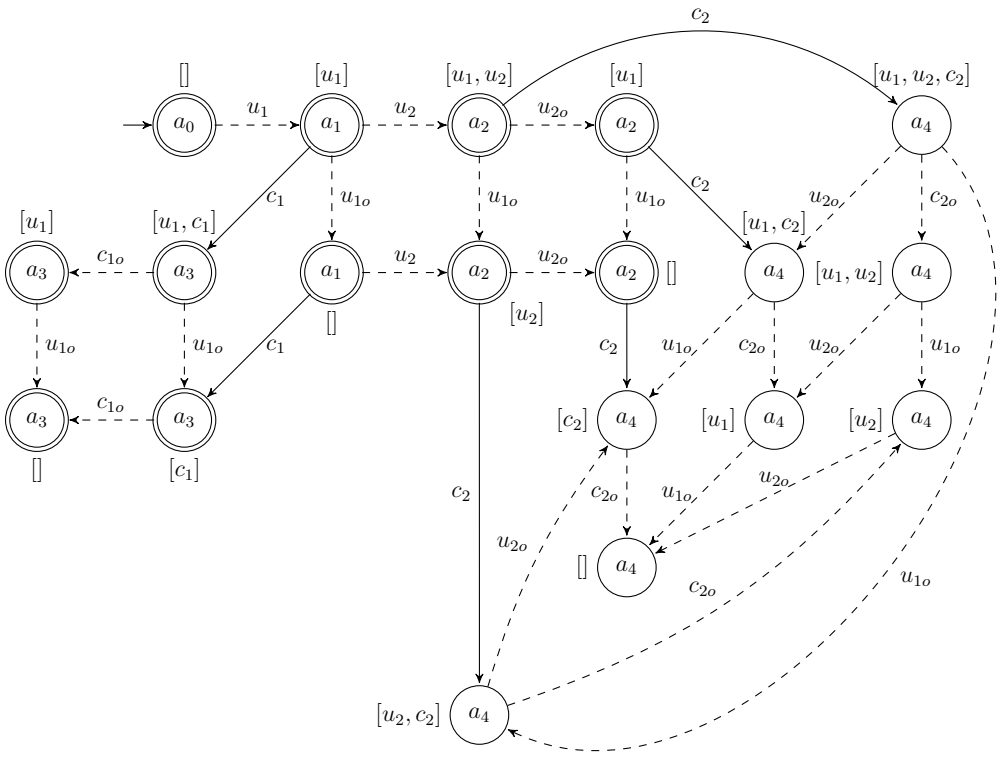


Figure 2.5: Observed plant for G from Example 2.1.

The next step is to determine all feasible enabling events that could have been sent in the asynchronous setting based on the observed plant. However, since enabling events are only related to controllable events, we leave out the uncontrollable events of the observed plant. We also use the plant model to determine how events will be executed in the plant, and how the observations can occur in the asynchronous plant. The asynchronous plant automaton is achieved using the operator given in Definition 2.9. To obtain a finite automaton, the size of observation and control channels are limited to N_o and N_c, respectively. These limitations are required to guarantee the finiteness of the set of states in the presence of an event-loop.

Definition 2.9 (Asynchronous Plant Operator)**.** Given a plant $G = (A, \Sigma, \delta, a_0, A_m)$ and constants N_c and N_o, Π gives the asynchronous plant as the following automaton:

$$\Pi(G, N_c, N_o) = (X, \Sigma_{ASP}, \delta_{AP}, x_0, X_m), \tag{2.2}$$

Let $OP' = P_{\Sigma_{OP} \setminus \Sigma_{uc}}(\Upsilon(G, N_o)) = (Q', \Sigma_{OP}, \delta_{OP'}, q'_0, Q'_m)$, and

$$\begin{aligned} X &= A \times Q' \times M \times L, \\ x_0 &= (a_0, q'_0, [], \varepsilon), \\ X_m &= A_m \times Q' \times M \times L. \end{aligned}$$

For $a \in A$, $q' \in Q'$, $m \in M$ and $l \in L$, the transition function $\delta_{AP} : X \times \Sigma_{ASP} \to X$ is defined as follows:

1. If $\delta_{OP'}(q', \sigma)!$, $\sigma \in \Sigma_c$

$$\delta_{AP}((a, q', m, l), \sigma_e) = (a, \delta_{OP'}(q', \sigma), m, app(l, \sigma)).$$

2. If $\delta(a, \sigma)!$, $head(l) = \sigma, \sigma \in \Sigma_c$

$$\delta_{AP}((a, q', m, l), \sigma) = (\delta(a, \sigma), q', m \uplus [\sigma], tail(l)).$$

3. If $\delta(a, \sigma)!, \sigma \in \Sigma_{uc}$

$$\delta_{AP}((a, q', m, l), \sigma) = (\delta(a, \sigma), q', m \uplus [\sigma], l).$$

4. If $\sigma \in m$, $\delta_{OP'}(q', \sigma_o)!$

$$\delta_{AP}((a, q', m, l), \sigma_o) = (a, \delta_{OP'}(q', \sigma_o), m \setminus [\sigma], l).$$

5. If $\sigma \in m$, $\neg\delta_{OP'}(q', \sigma_o)!$

$$\delta_{AP}((a, q', m, l), \sigma_o) = (a, q', m \setminus [\sigma], l).$$ ■

Property 2.2 (Finite Asynchronous Plant)**.** For a given plant G, which is being supervised and observed through the control and observation channels with limited capacities N_c and N_o, respectively, $\Pi(G, N_c, N_o)$ is a finite automaton.

Proof. See Appendix A.2.2. ■

Example 2.3. For the plant given in Example 2.1, we use the projected observed plant OP' depicted in Figure 2.6 to determine the occurrences of the enabling events in the asynchronous plant for which we do not need the state information of OP. The asynchronous plant is given in Figure 2.7. Each state indicates the current states of G and OP', and also the events existing in the control and observation channels, which are not shown in the figure due to lack of space. Note that the plant is free to execute a control command sent by the supervisor or ignore it. For instance, after the execution of $u_1\, u_2$, the event c_{1e} may occur. However, this enabling event is ignored as the event c_1 will never be executed.

The asynchronous plant determines all the feasible enabling events, executions of events in the plant, and observations of them. An asynchronous supervisor is then synthesized for the asynchronous plant to determine which of the enabling events need to be disabled. The synthesis algorithm is presented in Algorithm 2.1 in which we use the following additional concepts for some automaton AP with event set Σ_{ASP}:

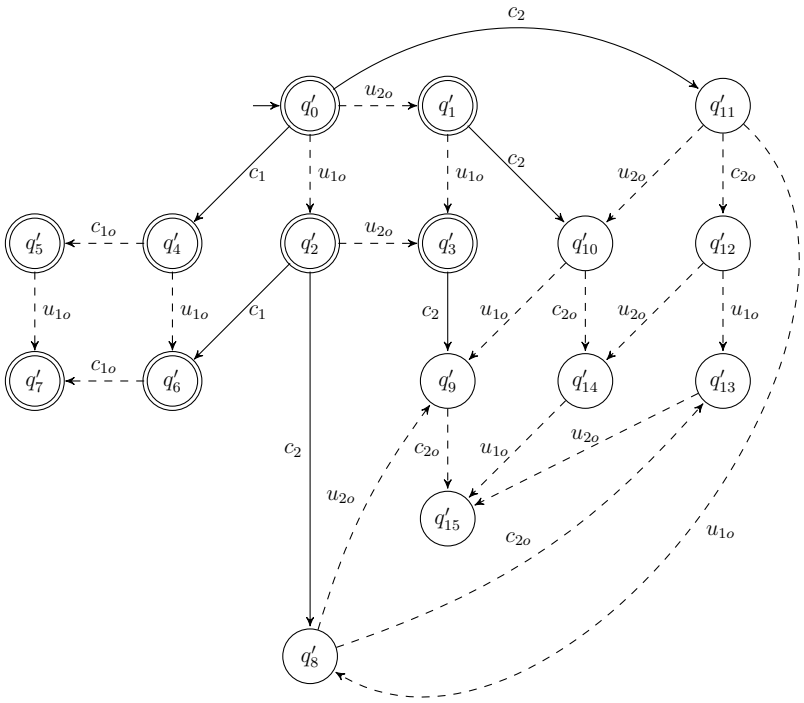


Figure 2.6: OP' for G from Example 2.1.

- *Blocking*(AP) is the set of blocking states in AP.
- For a set of states BS, $Uncon_{AP}(BS)$ is the smallest set of states such that
  1. $BS \subseteq Uncon_{AP}(BS)$;
  2. if $\delta_{AP}(x, \sigma) \in Uncon_{AP}(BS)$ for some $x \in X$ and $\sigma \in \Sigma \cup \Sigma_o$, then $x \in Uncon_{AP}(BS)$;

  Intuitively this set provides all the states from which a state from BS can be reached in an uncontrollable way.
- Events executed in the plant are unknown to a supervisor until it receives the observations. Therefore, while doing synthesis we need to be careful that the plant events are unobservable in the asynchronous plant, and a supervisor should make the same decision for any two words that it cannot distinguish between. To consider this issue in the synthesis algorithm, we use the function $OBS_{AP}(x) = \{x' \in X \mid \exists w, w' \in \Sigma^*_{AP}, \delta_{AP}(x_0, w) = x \wedge \delta_{AP}(x_0, w') = x' \wedge P_{\Sigma_e \cup \Sigma_o}(w) = P_{\Sigma_e \cup \Sigma_o}(w')\}$, which gives the set of states reachable through the same observation (observationally equivalent states). For instance, $OBS_{AP}(x_0) = \{x_0, x_1, x_2\}$ in Figure 2.7.

Additionally, we could also have assumed that some events from Σ_e are unobservable. In this case, there would be more states that become observationally equivalent, and so the resulting supervisor could be more restrictive since a control command should be disabled at all observationally equivalent states if it needs to be disabled at one of them.

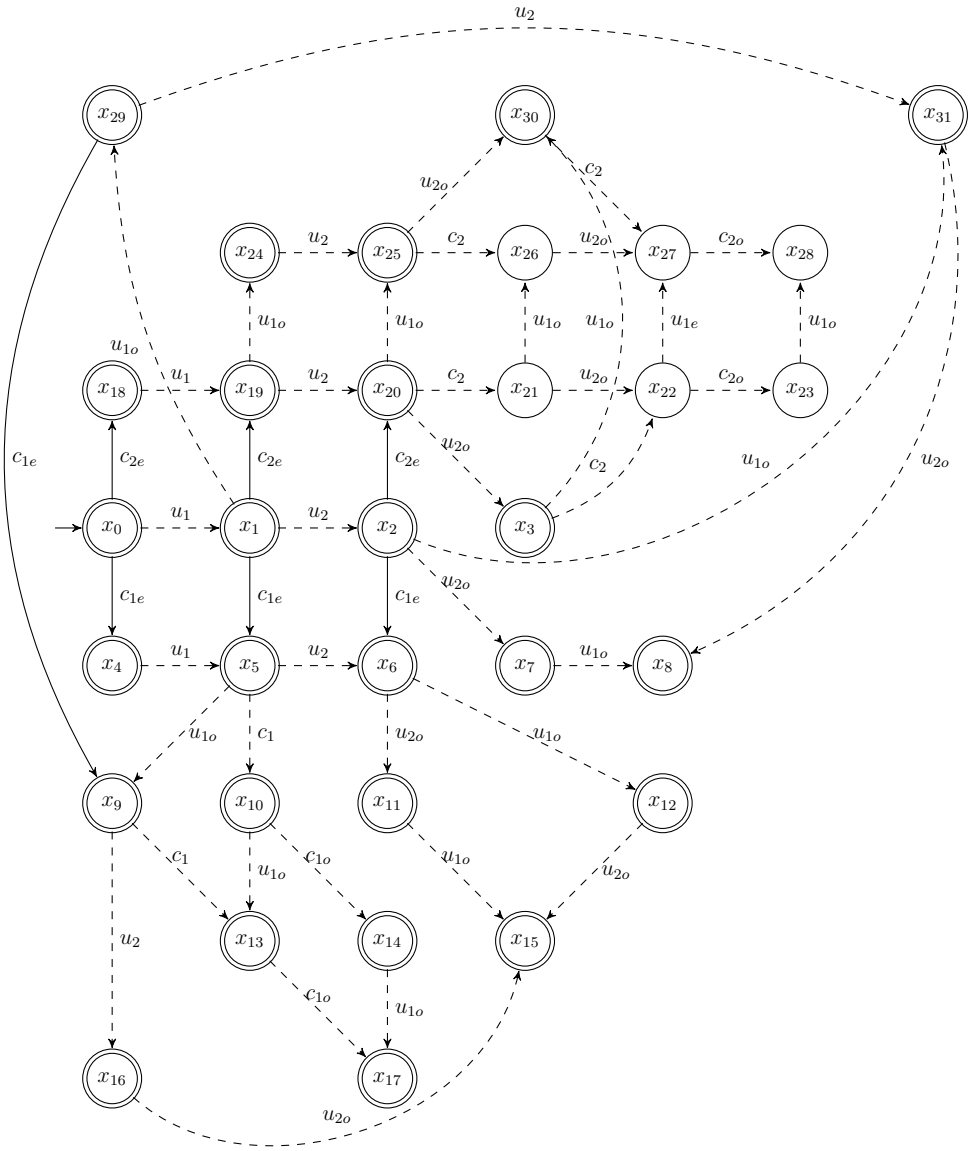


Figure 2.7: Asynchronous plant for G from Example 2.1.

Starting from $AS = AP$, Algorithm 2.1 changes AS at line 7 by disabling transitions labeled by enabling events that lead to states from $Uncon_{AP}(BS)$, and delivering the reachable part at line 11.

Example 2.4. Let us reconsider the plant from Example 2.1. By applying Algorithm 2.1, we achieve the asynchronous supervisor given in Figure 2.8. Note that if c_2 was replaced by c_1 in the plant (Figure 2.2a), then no result exists, precisely because in the asynchronous setting observation of u_2 is not immediate.

Note also that for any asynchronous supervisor resulting from Algorithm 2.1, the asynchronously supervised plant is a finite automaton. Moreover, as mentioned in Property 2.1, controllability of the synthesized asynchronous supervisor is already guaranteed.

Algorithm 2.1 Asynchronous supervisory control synthesis
Input: $AP = (X, \Sigma_{ASP}, \delta_{AP}, x_0, X_m)$, Σ_{uc}, Σ_c
Output: $AS = (Y, \Sigma_{AS}, \delta_{AS}, y_0, Y_m)$ or no result

1: $AS \leftarrow AP$
2: $BS \leftarrow Blocking(AS)$
3: **while** $x_0 \notin BS \wedge BS \neq \varnothing$ **do**
4: **for** $y \in Y \wedge \sigma \in \Sigma_e$ **do**
5: **if** $\delta_{AS}(y, \sigma) \in Uncon_{AS}(BS)$ **then**
6: **for** $y' \in OBS_{AS}(y)$ **do**
7: $\delta_{AS}(y', \sigma) \leftarrow$ **undefined**
8: **end for**
9: **end if**
10: **end for**
11: $AS \leftarrow Reach(AS)$
12: $BS \leftarrow Blocking(AS)$
13: **end while**
14: **if** $x_0 \in BS$ **then**
15: no result
16: **end if**
17: $AS \leftarrow P_{\Sigma_{ASP} \setminus \Sigma}(AS)$

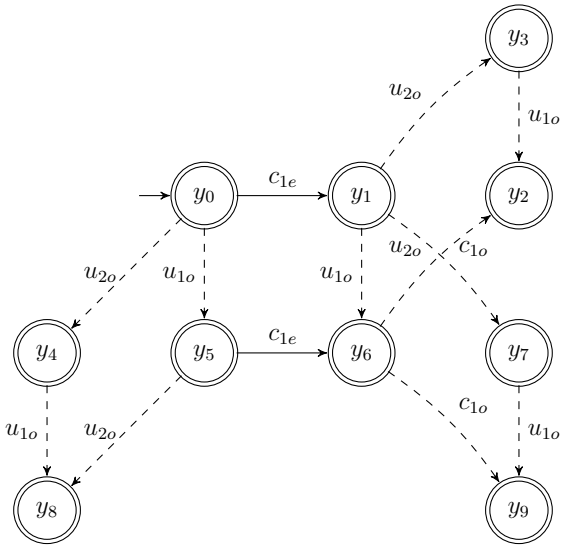


Figure 2.8: Asynchronous supervisor for G from Example 2.1.

In Theorem 2.1, we prove that the asynchronous supervisor obtained from Algorithm 2.1 guarantees nonblockingness.

Theorem 2.1 (Nonblocking Asynchronous Supervisor)**.** For $G = (A, \Sigma, \delta, a_0, A_m)$ and $AS = (Y, \Sigma_{AS}, \delta_{AS}, y_0, Y_m)$ achieved from Algorithm 2.1, $AS||G$ is nonblocking.

Proof. See Appendix A.2.3 ■

2.5 Conclusions

In this chapter, we investigate the problem of inexact synchronization that may cause a conventionally synthesized supervisor to fail in practice. For this purpose, we first present an asynchronous supervisory control setting in which control commands may arrive in the plant after some delay, and events executed in the plant may not be immediately observed. We also assume that events executed in the plant in some order could be observed in a different order. Moreover, we assume that the plant can execute controllable events only if they are commanded by the supervisor. On the other hand, the plant is free to execute a control command sent by the supervisor or ignore it. In the asynchronous setting, uncontrollable events may be executed in the plant spontaneously, and a supervisor has no role in the occurrence of them. Therefore, controllability is always guaranteed by the definition of asynchronous composition. Furthermore, we present a method for achieving an automaton called the asynchronous plant representing the behavior of the plant in the asynchronous setting. Finally, a synthesis algorithm is presented for obtaining an asynchronous supervisor guaranteeing nonblockingness. We solved the basic synthesis problem to provide nonblockingness. To synthesize an asynchronous supervisor satisfying control requirements, we can first translate the requirements to the plant, and then apply the basic synthesis technique.

Chapter 3

Networked Supervisory Control of Timed Discrete-Event Systems

Conventional supervisory control theory assumes full synchronization between the supervisor and the plant. This assumption is violated in a networked-based communication setting due to the presence of delays, and this may result in incorrect behavior of a supervisor obtained from conventional supervisory control theory. This chapter presents a technique to synthesize a networked supervisor handling communication delays. For this purpose, first, a networked supervisory control framework is provided, where the supervisor interacts with the plant through control and observation channels, both of which introduce delays. The control channel is FIFO, but the observation channel is assumed to be non-FIFO so that the events may not necessarily be observed by the supervisor in the same order as they occurred in the plant. It is assumed that a global clock exists in the networked control system, and so the communication delays are represented in terms of clock *ticks*. Based on the proposed framework, a networked plant automaton is achieved, which models the behavior of the plant under the effects of communication delays and disordered observations. Based on the networked plant, the networked supervisor is synthesized, which is guaranteed to be (timed networked) controllable, nonblocking, time-lock free, (timed networked) maximally permissive, and satisfies control requirements for the plant.

This chapter is based on Rashidinejad et al. [2021a] with an earlier version as Rashidinejad et al. [2018].

3.1 Introduction

Networked control of systems has gained a lot of attention in recent years. By eliminating unnecessary wiring, the cost and complexity of a control system are reduced, and nodes can more easily be added to or removed from the system. More importantly, there are applications in which the system is required to be controlled over a distance such as telerobotics, space explorations, and working in hazardous environments [Gupta and Chow 2010].

Networked control of systems is challenging due to network communication problems among which delays have the highest impact [Heemels et al. 2010]. In this regard, many works have appeared in the literature investigating the effects of communication delays on the performance of a control system with time-based dynamics [Antsaklis and Baillieul 2007; Gupta and Chow 2010; Heemels et al. 2010]. However, there is less work considering networked control of discrete-event systems (DESs).

A DES consists of a set of discrete states where state transitions depend only on the occurrence of instantaneous events. DESs are used for modeling many types of systems, e.g., manufacturing processes, traffic, and queuing systems [Cassandras and Lafortune 2009]. In DESs, time is typically neglected meaning that events can occur independently of time. However, there are control applications in which time is an important factor to be considered, such as minimizing the production-cycle time in a manufacturing process [Wonham 2015]. To consider time in control of a DES, the concept of a timed discrete-event system (TDES) has been introduced, in which the passage of a unit of time is indicated by an event called *tick* [Brandin and Wonham 1994].

Supervisory control theory is the main control approach developed for DESs [Ramadge and Wonham 1987]. To achieve desired (safe) behavior, a supervisor observes events executed in the plant and determines which of the next possible events must be disabled. Supervisory control theory synthesizes nonblocking supervisors that ensure safety, controllability, and nonblockingness for the plant and do not unnecessarily restrict the behavior of the plant (maximal permissiveness) [Cassandras and Lafortune 2009].

In conventional supervisory control theory [Cassandras and Lafortune 2009; Ramadge and Wonham 1987], the plant generates all events, while the supervisor can disable some of the events and observes synchronously the execution of events in the plant. Based on this synchronous interaction, a model of the controlled plant behavior can be obtained by synchronous composition of the respective models of the plant and the supervisor. However, the synchronous interaction assumption fails in a networked supervisory control setting, due to the presence of delays in the communication channels between the plant and supervisor.

There are several works in the literature investigating supervisory control of DES under communication delays. There are three important properties that these works may focus on:

- Nonblockingness. For many applications, it is important to guarantee that the supervised plant does not block (as an additional control requirement) [Cassandras and Lafortune 2009; Xu et al. 2017; Yin and Lafortune 2015].

- Maximal permissiveness. A supervisor must not restrict the plant behavior more than necessary so that the maximal admissible behavior of the plant is preserved [Cassandras and Lafortune 2009; Wonham 2015].
- Modeling delays based on time. In most of the existing approaches such as in Balemi [1992], Lin [2014], Liu et al. [2019], Park [2012], Park and Cho [2006], Rashidinejad et al. [2019a], Shu and Lin [2017a], and Xu et al. [2017], communication delays are measured in terms of a number of consecutive event occurrences. As stated in Rashidinejad et al. [2018], Shu and Lin [2017a], and Zhao et al. [2017], it is not proper to measure time delay only based on the number of event occurrences since events may have different execution times. Here, as in TDES [Wonham 2015], the event *tick* is used to represent the passage of a unit of time, which is the temporal resolution for modeling purposes.

Supervisory control synthesis under communication delays was first investigated by Balemi [Balemi 1992]. To solve the problem, Balemi defines a condition called *delay insensitive language*. A plant has a delay insensitive language whenever any control command, enabled at a state of the plant, is not invalidated by an uncontrollable event. In other words, any control command sent by the supervisor is acceptable by the plant both before and after the uncontrollable event. Under this condition, supervisory control under communication delays can be reduced to the conventional supervisory control synthesis [Balemi 1992]. In other words, if a given plant has a delay insensitive language, then the conventional supervisor is robust to the effects of delays. The benefit of this method is that nonblockingness and maximal permissiveness are already guaranteed by the supervisor if it exists (as they are guaranteed in the conventional supervisory control theory). However, the imposed condition restricts the applications for which such a supervisor exists.

In Park [2012] and Park and Cho [2006], a condition called *delay observability* is defined for the control requirement. If the control requirement is delay observable, then a networked supervisor exists. The delay observability condition is similar to the delay insensitivity condition generalized for a sequence of uncontrollable events so that a control command is not invalidated by a sequence of consecutive uncontrollable events. In Park [2012] and Park and Cho [2006], nonblockingness is guaranteed. However, maximal permissiveness is not guaranteed. Also, no method is proposed to obtain the supremal controllable and delay observable sublanguage of a given control requirement [Park 2012].

In a more recent study, Lin introduced new observability and controllability conditions considering the effects of communication delays called *network controllability* and *network observability* [Lin 2014]. The approach presented by Lin has been further modified in [Alves et al. 2017; Lin 2014; Shu and Lin 2015, 2017a,b; Xu et al. 2017; Zhao et al. 2017]. In all these works, the problem of supervisory control synthesis under communication delays is defined under certain conditions (network controllability and network observability or the modified versions of them). When the conditions are not met (by the control requirement), it is not guaranteed to have a (networked) supervisor [Alves et al. 2017; Lin 2014; Shu and Lin 2015; Xu et al. 2017; Zhao et al. 2017]. As discussed in Shu and Lin [2017a], delayed observations and delayed control commands make it (more) challenging to ensure

nonblockingness of the supervised plant (compared to the conventional non-networked setting when there is no delay). To guarantee nonblockingness, additional conditions are imposed on the control requirement in Xu et al. [2017], but maximal permissiveness is not investigated.

In Liu et al. [2019], an online predictive supervisory control synthesis method is presented to deal with control delays. The supervisor is claimed to be maximally permissive. However, this is not formally proved. This is also the case in Shu and Lin [2015] as they do not formally prove the maximal permissiveness although they establish the steps to achieve it. In Shu and Lin [2017b], a predictive synthesis approach is proposed to achieve a networked supervisor that is guaranteed to be maximally permissive if it satisfies the conditions. Nonblockingness is yet not investigated in Shu and Lin [2017b]. None of the works following Lin's method consider simultaneously nonblockingness and maximal permissiveness. Moreover, as discussed in a recent study by Lin, in the case that the conditions are not met by the control requirement, there is no method so far to compute the supremal sublanguage satisfying the conditions [Lin 2020].

In Chapter 2 and in Rashidinejad et al. [2018], a new synthesis algorithm is proposed in which the effects of communication delays are taken into account in the synthesis procedure instead of in extra conditions to be satisfied by the plant/control requirement.

The asynchronous setting, proposed in Chapter 2, does not take time into account, but it is guaranteed that the synthesized (asynchronous) supervisor satisfies nonblockingness. Maximal permissiveness is still an open issue.

The networked supervisory control setting, presented in this chapter, is close to the asynchronous supervisory control framework introduced in Chapter 2. We use the same notations for enabling, execution, and observation of events. However, the networked supervisory control setting presented here is different from the asynchronous supervisory control technique presented in Chapter 2 mainly because here we quantify the amount of delay. Rashidinejad et al. [2018] focuses on modeling delays based on time, but it does not formally prove nonblockingness or maximal permissiveness.

In Zhu et al. [2019b] as a more recent study, first, the control and observation channels are modeled. Then, both the plant and control requirements are transformed into a networked setting. Using these transformations, the problem of networked supervisory control synthesis is reduced to conventional supervisory control synthesis. Using conventional supervisory control synthesis, the resulting supervisor is controllable and nonblocking for the transformed plant and the transformed control requirements. However, it is not discussed if the supervisor satisfies these conditions for the (original) plant.

Furthermore, although it is important to consider time in the presence of delays, only a few papers investigate networked supervisory control of TDES [Alves et al. 2017; Miao et al. 2019; Park and Cho 2008; Rashidinejad et al. 2018; Zhao et al. 2017] (where communication delays are modeled based on a consistent unit of time) as it introduces new complexities and challenges.

Table 3.1 gives an overview of the existing works. To the best of our knowledge, none of these works study supervisory control synthesis of discrete-event systems under

communication delays such that delays are modeled based on time, and the delivered supervisor guarantees both nonblockingness and maximal permissiveness as is done here.

	Synthesis Method	**Nonblocking**	**Permissive**	**Delays**
Balemi [1992]	conventional supervisory control for delay-insensitive plant	✓	✓	event-based
Park [2012] and Park and Cho [2006]	conventional supervisory control for delay-observable requirement	✓	✗	event-based
Lin [2014], Liu et al. [2019], and Shu and Lin [2015]	networked supervisory control for network controllable and network observable requirement	✗	✗	event-based
Xu et al. [2017]	networked supervisory control for network controllable and network observable requirement	✓	✗	event-based
Rashidinejad et al. [2019a]	new algorithm under the effects of delays	✓	✗	event-based
Shu and Lin [2017b]	networked supervisory control for network controllable and network observable requirement	✗	✓	event-based
Park and Cho [2008]	conventional supervisory control for delay-nonconflicting requirement	✗	✗	time-based
Alves et al. [2017], Miao et al. [2019], and Zhao et al. [2017]	networked supervisory control for network controllable and network observable requirement	✗	✗	time-based
Rashidinejad et al. [2018]	new algorithm under the effects of delays	✗	✗	time-based
Our Approach	new algorithm under the effects of delays	✓	✓	time-based

Table 3.1: Overview of existing works.

Our work is close to Rashidinejad et al. [2018] in terms of the networked supervisory control setting and to Chapter 2 in terms of the synthesis technique. Similar to Rashidinejad et al. [2018] and Chapter 2, the following practical conditions are taken into account:

1. A controllable event can be executed in the plant only if it is commanded (enabled) by the supervisor.
2. An uncontrollable event is not commanded (enabled) by the supervisor; it occurs spontaneously in the plant.
3. Any event, controllable or uncontrollable, executed in the plant is observable to the supervisor.
4. A control command sent by the supervisor reaches the plant after a constant amount of time delay. The command may not necessarily be accepted by the plant, in which case it will be removed from the control channel when the next *tick* occurs. Also, the observation of a plant event, controllable or uncontrollable, occurs after a constant amount of time delay. See Section 3.3.3 for discussions on how the proposed solution is adjusted for situations where 1) delays are specified to events (specific amounts of control and observation delays are specified to each event), and 2) delays are bounded (not constant).
5. The control channel is assumed to be FIFO, so control commands sent by the supervisor will reach the plant in the same order as they have been sent. However,

the observation channel is non-FIFO [2], and so consecutive events that occur in the plant may be observed by the supervisor in any possible order. For instance, if the events a and b occur in that order between two *tick*s in the plant, they may be observed in the other order. Here, we investigate the situation where only the observation channel is non-FIFO. See Section 3.3.3 for a discussion on how the proposed solution is adapted for a non-FIFO control channel.

This chapter differs from Chapter 2 and Rashidinejad et al. [2018] in the following aspects:

1. Modeling purposes. In [Wonham 2015], a TDES is a DES in which the execution of each event, called an *active event* transition, is restricted within a lower and an upper time bound specified to the event. The time bounds are given as natural numbers. From such a TDES, a *timed transition graph* (TTG) is derived based on the active event transitions in the DES and the given time bounds for each event. At the end, a TTG is just a transition graph that explicitly includes transitions labeled by the event *tick* as well as active event transitions. The kind of TDES considered in this paper (see Section 3.2) is a TTG that is formally represented by an automaton. Unlike TDES proposed by Wonham in [Wonham 2015], there is no specific relationship between the occurrences of *tick* and other events in the TDES of this paper. As a result, the TDES in this paper includes the one proposed by Wonham in [Wonham 2015] and is a more general modeling framework than that. To take the time progress of the system into account, the concept of time-lock freeness is introduced as a property for a TDES. Time-lock freeness, similar to nonblockingness, is guaranteed by the networked supervisor.
2. Synthesis technique. Inspired from the idea introduced in Chapter 2 to synthesize an asynchronous supervisor for DES, the synthesis method proposed in Rashidinejad et al. [2018] for networked supervisory control of TDES is improved. For this purpose, first, the networked supervisory control (NSC) framework is modeled. Then, a networked plant automaton is proposed, modeling the behavior of the plant in the NSC framework. Based on the networked plant, a networked supervisor is synthesized. It is guaranteed that the networked supervisor provides nonblockingness, time-lock freeness, and maximal permissiveness.
3. Control requirement. The control requirement in Chapter 2 and in Rashidinejad et al. [2018] is limited to the avoidance of illegal states. Here, the networked supervisory control synthesis is generalized to control requirements modeled as automata.

The rest of the chapter is organized as follows. The NSC framework is introduced in Section 3.2. For the NSC framework, an operator is proposed to give the networked supervised plant. Moreover, the conventional controllability and maximal permissiveness conditions are modified to timed networked controllability and timed networked maximal permissiveness conditions suitable for the NSC framework. Then, the basic networked supervisory control synthesis problem is formulated, which aims to find a timed networked controllable and timed networked maximally permissive networked supervisor guaranteeing

[2] Non-FIFO observation is related to the interleave sensitivity problem from Chapter 2.

nonblockingness and time-lock freeness of the networked supervised plant. In Section 3.3, first, the networked plant is defined as an automaton representing the behavior of the plant under communication delays and disordered observations. Furthermore, a technique is presented to synthesize a networked supervisor that is a solution to the basic networked supervisory control problem. In Section 3.4, the basic networked supervisory control synthesis problem is generalized to satisfy a given set of control requirements. Relevant examples are provided in each section. Finally, Section 3.5 concludes this chapter.

3.2 Basic NSC Problem

3.2.1 Conventional Supervisory Control Synthesis of TDES

A TDES T is formally represented as a quintuple

$$T = (A, \Sigma, \delta, a_0, A_m),$$

where A, Σ, $\delta : A \times \Sigma \to A$, $a_0 \in A$, and $A_m \subseteq A$ stand for the set of states, the set of events, the (partial) transition function, the initial state, and the set of marked states, respectively. The set of events of any TDES is assumed to contain the event $tick \in \Sigma$. The set $\Sigma_a = \Sigma \setminus \{tick\}$ is called the set of active events. The notation $\delta(a, \sigma)!$ denotes that δ is defined for state a and event σ, i.e., there is a transition from state a with label σ to some state. The transition function is generalized to words in the usual way: $\delta(a, w) = a'$ means that there is a sequence of subsequent transitions from state a to the state a' that together make up the word $w \in \Sigma^*$. Starting from the initial state, the set of all possible words that may occur in T is called the language of T and is denoted by $L(T)$; $L(T) := \{w \in \Sigma^* \mid \delta(a_0, w)!\}$. Furthermore, for any state $a \in A$, the function $Reach(a)$ gives the set of states reachable from the state a; $Reach(a) := \{a' \in A \mid \exists w \in \Sigma^*, \delta(a, w) = a'\}$. States from which it is possible to reach a marked state are called nonblocking. An automaton is nonblocking when each state reachable from the initial state is nonblocking; for each $a \in Reach(a_0)$, $Reach(a) \cap A_m \neq \varnothing$. $L_m(T)$ denotes the marked language of T; $L_m(T) := \{w \in L(T) \mid \delta(a_0, w) \in A_m\}$. States from which time can progress are called *time-lock free* (TLF). An automaton is TLF when each state reachable from the initial state is TLF; for each $a \in Reach(a_0)$, there exists a $w \in \Sigma^*$ such that $\delta(a, w\, tick)!$.

Definition 3.1 (Natural Projection [Cassandras and Lafortune 2009])**.** For sets of events Σ and $\Sigma' \subseteq \Sigma$, $P_{\Sigma'} : \Sigma^* \to \Sigma'^*$ is defined as follows: for $e \in \Sigma$ and $w \in \Sigma^*$,

$$P_{\Sigma'}(\epsilon) := \epsilon,$$
$$P_{\Sigma'}(we) := \begin{cases} P_{\Sigma'}(w)e & \text{if } e \in \Sigma', \\ P_{\Sigma'}(w) & \text{if } e \in \Sigma \setminus \Sigma'. \end{cases}$$

The definition of natural projection is extended to a language $L \subseteq \Sigma^*$; $P_{\Sigma'}(L) := \{w' \in \Sigma'^* \mid \exists w \in L, P_{\Sigma'}(w) = w'\}$ [Cassandras and Lafortune 2009]. ■

Natural projection is an operation that is generally defined for languages. However, it is also possible to apply it on automata [Ware and Malik 2008]. For an automaton

with event set Σ, $P_{\Sigma'}$ first replaces all events not from Σ' by the silent event τ. Then, using a determinization algorithm (such as the one introduced in Hopcroft et al. [2006]), the resulting automaton is made deterministic. A state of a projected automaton is then marked if it contains at least one marked state from the original automaton (see [Hopcroft et al. 2006] for more details). Using the notation δ_P for the transition function of the projected automaton, we state the following properties of this construction: (1) for any $w \in \Sigma^*$, if $\delta(a_0, w) = a_r$ then $\delta_P(A_0, P_{\Sigma'}(w)) = A_r$ where A_0 is the initial state of the projected automaton, and $A_r \subseteq A$ is a set with $a_r \in A_r$, (2) for any $w \in \Sigma^*$, if $\delta(a_0, w) \in A_m$, then $\delta_P(A_0, P_{\Sigma'}(w))$ is a marked state in the projected automaton.

In the rest of the chapter, the plant is given as the TDES G represented by the automaton $(A, \Sigma_G, \delta_G, a_0, A_m)$ with $\Sigma_G = \Sigma_a \cup \{tick\}$ and $\Sigma_a \cap \{tick\} = \varnothing$. Also, as it holds for many applications, G is a finite automaton [Wonham 2015]. A finite automaton has a finite set of states and a finite set of events [Carroll and Long 1989].

Here, it is assumed that all events in G are observable. A subset of the active events $\Sigma_{uc} \subseteq \Sigma_a$ is uncontrollable. $\Sigma_c = \Sigma_a \setminus \Sigma_{uc}$ gives the set of controllable active events. The event *tick* is uncontrollable by nature. However, as in Brandin and Wonham [1994], it is assumed that *tick* can be preempted by a set of forcible events $\Sigma_{for} \subseteq \Sigma_a$. Note that forcible events can be either controllable or uncontrollable. For instance, closing a valve to prevent overflow of a tank, and the landing of a plane are controllable and uncontrollable forcible events, respectively [Wonham 2015]. Note that for synthesis, the status of the event *tick* lies between controllable and uncontrollable depending on the presence of enabled forcible events. To clarify, when the event *tick* is enabled at some state a and also there exists a forcible event $\sigma \in \Sigma_{for}$ such that $\delta_G(a, \sigma)!$, then *tick* is considered as a controllable event since it can be preempted. Otherwise, *tick* is an uncontrollable event. In the figures, forcible events are underlined. The transitions labelled by controllable (active or *tick*) events are indicated by solid lines and the transitions labelled by uncontrollable (active or *tick*) events are indicated by dashed lines.

If the plant G is blocking, then a supervisor S needs to be synthesized to satisfy nonblockingness of the supervised plant. S is also a TDES with the same event set as G. Since the plant and supervisor are supposed to work synchronously in a conventional non-networked setting, the automaton representing the supervised plant behavior is obtained by applying the *synchronous product* denoted by $S||G$ [Cassandras and Lafortune 2009]. Generally, in the synchronous product of two automata, a shared event can be executed only when it is enabled in both automata, and a non-shared event can be executed if it is enabled in the corresponding automaton. Since the conventional supervisor S has the same event set as G, each event will be executed in $S||G$ only if the supervisor enables (allows) it. S is controllable if it allows all uncontrollable events that may occur in the plant. This is captured in *conventional controllability for TDES*, which is reformulated from [Wonham 2015].

Definition 3.2 (Conventional Controllability for TDES)**.** Given a plant G with uncontrollable events Σ_{uc} and forcible events Σ_{for}, a TDES S, is controllable w.r.t. G if for all $w \in L(S||G)$ and $\sigma \in \Sigma_{uc} \cup \{tick\}$, if $w\sigma \in L(G)$,

1. $w\sigma \in L(S||G)$, or

2. $\sigma = tick$ and $w\sigma_f \in L(S||G)$ for some $\sigma_f \in \Sigma_{for}$. ■

Item (1) in the above definition is the standard controllability property (when there is no forcible event to preempt *tick*); S cannot disable uncontrollable events that G may generate. However, if a forcible event is enabled, this may preempt the time event, which is captured by item (2).

A supervisor S is called *proper* for a plant G whenever S is controllable w.r.t. G, and the supervised plant $S||G$ is nonblocking.

Definition 3.3 (Conventional Maximal Permissivenesss)**.** A proper supervisor S is *maximally permissive* for a plant G, whenever S preserves the largest behavior of G compared to any other proper supervisor S'; for any proper S': $L(S'||G) \subseteq L(S||G)$. ■

For a TDES, a proper and a maximally permissive supervisor can be synthesized by applying the synthesis algorithm proposed in Wonham [2015].

3.2.2 Motivating Examples

This section discusses the situations where a proper and maximally permissive conventional supervisor S fails in the presence of observation delay (Example 3.1), non-FIFO observation (Example 3.2), or control delay (Example 3.3).

Example 3.1 (Observation Delay)**.** Consider the plant depicted in Figure 3.1. To be maximally permissive, S must not disable a at a_0, and to be nonblocking, S must disable a at a_2. Now, assume that the observation of the events executed in G are not immediately received by S due to observation delay. Starting from a_0, imagine that u occurs, and G goes to a_2. Since S does not observe u immediately, it supposes that G is still at a_0 where it enables a. Then, a will be applied at the real state where G is, i.e., a_2, and so G goes to a_3, which is blocking.

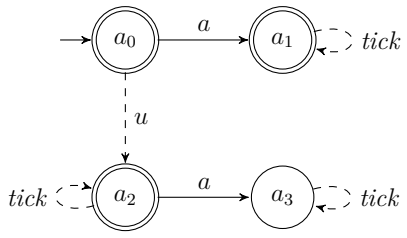


Figure 3.1: Plant for Example 3.1.

Example 3.2 (Non-FIFO Observation)**.** Consider the plant G depicted in Figure 3.2. To be nonblocking, S must disable a at a_3, and to be maximally permissive, S must not disable a at a_6. Now, assume that the observation channel is non-FIFO, i.e., events may be observed in a different order as they occurred in G. Starting from a_0, imagine that G executes $tick\,a\,b$ and goes to a_3. Since the observation channel is non-FIFO, S may receive the observation of $tick\,a\,b$ as $tick\,b\,a$ after which it does not disable a. However, G is actually at a_3 and by executing a, it goes to a_4, which is blocking.

Example 3.3 (Control Delay)**.** Consider the plant depicted in Figure 3.3. To be maximally permissive, S must not disable a at a_1, and to be nonblocking S must disable a at a_3. Now, assume that control commands are received by G after one *tick*. Starting from a_0, S

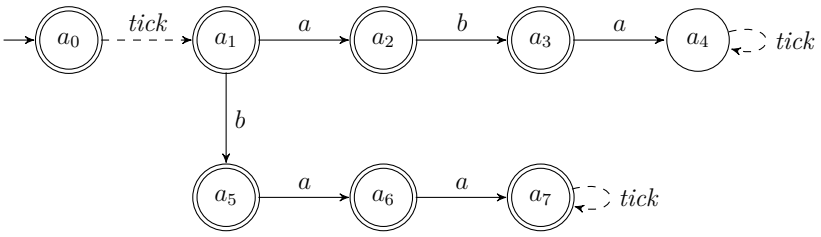


Figure 3.2: Plant for Example 3.2.

does not disable a after one *tick* (when G is at a_1). However, the command is received by G after the passage of one *tick* (due to the control delay) when G is at a_3. So, by executing a at a_3, G goes to a_4, which is blocking.

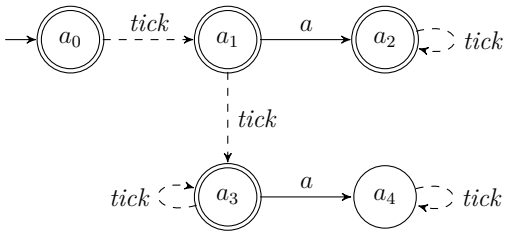


Figure 3.3: Plant for Example 3.3.

Remark. Conventional supervisory control synthesis of a TDES guarantees nonblockingness [Wonham 2015]. However, as can be seen in Example 3.2, it cannot guarantee time-lock freeness; a_3 is not TLF, and it is not removed by S. This is not an issue in Wonham [2015] since a TDES is assumed to satisfy the ALF condition. Here, to guarantee time progress, the TLF property must be considered in synthesis.

As is clear from the examples, a supervisor is required that can deal with the problems caused by communication delays and disordered observations. To achieve such a supervisor, first, the networked supervisory control framework is established.

3.2.3 NSC Framework

In the presence of delays in the control and observation channels, enabling, executing and observing events do not happen at the same time. Figure 3.4 depicts the networked supervisory control (NSC) framework that is introduced in this chapter. Figure 3.4 is inspired from the representation of the plant, supervisor, and the transmitted data between them in [Cassandras and Lafortune 2009]. To recognize the differences between the enablement and observation of events and their execution in the plant, as in Chapter 2 and in Rashidinejad et al. [2018], a set of *enabling events* Σ_e and a set of *observed events* Σ_o are introduced.

Definition 3.4 (Enabling and Observed Events)**.** Given a plant G, to each controllable active event $\sigma \in \Sigma_c$ an enabling event $\sigma_e \in \Sigma_e$, and to each active event $\sigma \in \Sigma_a$ an observed event $\sigma_o \in \Sigma_o$ are associated such that $\Sigma_e \cap \Sigma_a = \varnothing$ and $\Sigma_o \cap \Sigma_a = \varnothing$ (clearly $\Sigma_e \cap \Sigma_o = \varnothing$). ■

Note that all events executed in the plant are supposed to be observable so that the observed event σ_o is associated to any $\sigma \in \Sigma_a$. However, not all the events are supposed to be controllable. Uncontrollable events such as disturbances or faults occur in the plant spontaneously. In this regard, enabling events σ_e are associated only to events from Σ_c.

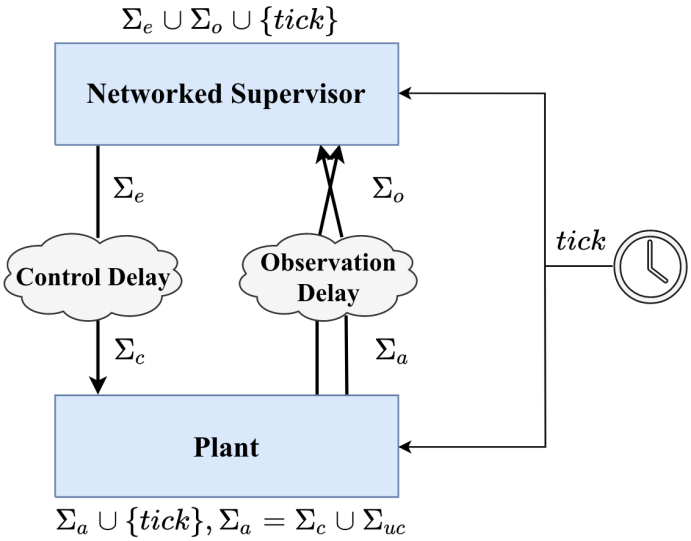


Figure 3.4: NSC framework.

Considering Figure 3.4, a networked supervisor for G that fits in the proposed framework is a TDES given as:

$$NS = (Y, \Sigma_{NS}, \delta_{NS}, y_0, Y_m),$$

for which the event set $\Sigma_{NS} = \Sigma_e \cup \Sigma_o \cup \{tick\}$, and the event *tick* is produced by the global clock in the system so that $\Sigma_{NS} \cap \Sigma_G = \{tick\}$.

For the proposed NSC framework, the behavior of the plant under the control of a networked supervisor is achieved through *timed asynchronous composition operator*. This operator is inspired from the asynchronous composition operator introduced in Chapter 2 and in Rashidinejad et al. [2018], where similar representations are proposed for observation and control channels.

To define timed asynchronous composition, we first need to consider the effects of delays on events sent through the control and observation channels. It is assumed that the control (observation) channel has a finite capacity denoted by L_{max} (M_{max}), which introduces a constant amount of delay represented by a natural number N_c (N_o). Since the control channel is supposed to be FIFO, a list or sequence is used to consider the journey of events through the control channel. As given in Definition 3.5 below, $l \in (\Sigma_c \times [0, N_c])^*$ provides us with the current situation of the control channel. The interpretation of $l[i] = (\sigma, n)$ is that the i^{th} enabling event present in the control channel is σ_e, which still requires n *ticks* before being received by the plant.

Definition 3.5 (Control Channel Representation)**.** The control channel is represented by the set $L = \{l \in (\Sigma_c \times [0, N_c])^* \mid |l| \leq L_{max}\}$. Moreover, we define the following operations for all $\sigma \in \Sigma_c$, the time counter $n \in [0, N_c]$ and $l \in L$:

- ε denotes the empty sequence.
- $app(l, (\sigma, n))$ adds the element (σ, n) to the end of l if $|l| < L_{max}$ (the channel is not full), otherwise l stays the same.
- $head(l)$ gives the first element of l (for nonempty lists). Formally, $head((\sigma, n)\ l)$ gives (σ, n) and $head(\varepsilon)$ is undefined.
- $tail(l)$ gives the list after removal of its leftmost element. Formally, $tail((\sigma, n)\ l)$ gives l, and $tail(\varepsilon)$ is undefined.
- $l - 1$ decreases the natural number component of every element in l by one (if possible). It is defined inductively as follows $\varepsilon - 1 = \varepsilon$, $((\sigma, 0)\ l) - 1 = l - 1$ removes the element with integer 0, and $((\sigma, n + 1)\ l) - 1 = (\sigma, n)\ (l - 1)$. ■

Due to the assumption that the observation channel is non-FIFO, we use a multiset to consider the journey of each event through the observation channel. As given in Definition 3.6 below, the multiset $m : \Sigma_a \times [0, N_o] \to \mathbb{N}$ is a total function that provides us with the current situation of the observation channel. The interpretation of $m(\sigma, n) = k$ is that currently there are k events σ in the observation channel that still require n *ticks* before reaching the (networked) supervisor.

Definition 3.6 (Observation Channel Representation)**.** The observation channel is represented by the set $M = \{m \,|\, m : \Sigma_a \times [0, N_o] \to \mathbb{N} \mid |m| \leq M_{max}\}$, where each element $m \in M$ is a multiset. Moreover, we define the following operations for all $m \in M$, $\sigma, \sigma' \in \Sigma_a$ and the time counters $n, n' \in [0, N_o]$:

- $[]$ denotes the empty multiset, i.e., the function m with $m(\sigma, n) = 0$. Often, a multiset is represented by enumerating the elements that occur in it a positive number of times. For instance, assuming an event set containing only events σ, σ' and σ'', a multiset m with $m(\sigma, n) = 2$, $m(\sigma', n') = 1$ and $m(\sigma'', n'') = 0$ is represented as $[(\sigma, n), (\sigma, n), (\sigma', n')]$. Also note that the ordering of elements in a multiset is immaterial, e.g., $[(\sigma, n), (\sigma', n), (\sigma, n)]$ denotes the same multiset.
- $|m| = \sum_{(\sigma,n)\in\Sigma_a\times[0,N_o]} m(\sigma, n)$ denotes the number of events in the observation channel represented by m.
- $m \uplus [(\sigma, n)]$ inserts (σ, n) to m if $|m| < M_{max}$ (the observation channel is not full). Formally, it denotes the function m' for which $m'(\sigma, n) = m(\sigma, n) + 1$ and $m'(\sigma', n') = m(\sigma', n')$ otherwise. If $|m| = M_{max}$ (the observation channel is full), then the channel stays the same, i.e., $m' = m$.
- $m \setminus [(\sigma, n)]$ removes (σ, n) from m once. Formally, it denotes the function m' for which $m'(\sigma, n) = \max(m(\sigma, n) - 1, 0)$ and $m'(\sigma', n') = m(\sigma', n')$ otherwise.
- $m - 1$ decreases the integer number component of every element by one as long as it is positive. Formally, it denotes the function m' for which $m'(\sigma, n) = m(\sigma, n + 1)$ for all $n < N_o$ and $m'(\sigma, N_o) = 0$. Note that as a consequence $[] - 1 = []$ and $[(\sigma, 0)] - 1 = []$.
- $(\sigma, n) \in m$ denotes that the pair (σ, n) exists in m, it holds if $m(\sigma, n) > 0$. ■

In the rest of the chapter, a networked supervisor for the plant G is given as the TDES NS represented by the automaton $(Y, \Sigma_{NS}, \delta_{NS}, y_0, Y_m)$.

Considering the representation of control and observation channels, a timed asynchronous composition operator is defined to achieve a networked supervised plant.

Definition 3.7 (Timed Asynchronous Composition Operator)**.** Given a plant G and a networked supervisor NS (for G), the asynchronous product of G and NS, denoted by $NS_{N_c}\|_{N_o} G$, is given by the automaton

$$NS_{N_c}\|_{N_o} G = (Z, \Sigma_{NSP}, \delta_{NSP}, z_0, Z_m),$$

where

$$Z = A \times Y \times M \times L, \qquad \Sigma_{NSP} = \Sigma_{NS} \cup \Sigma,$$
$$z_0 = (a_0, y_0, [], \varepsilon), \qquad Z_m = A_m \times Y_m \times M \times L.$$

Moreover, for $a \in A$, $y \in Y$, $m \in M$, and $l \in L$, $\delta_{NSP} : Z \times \Sigma_{NSP} \to Z$ is defined only in the following cases:

1. If $\delta_{NS}(y, \sigma_e)!$:

   $$\delta_{NSP}((a, y, m, l), \sigma_e) = (a, \delta_{NS}(y, \sigma_e), m, app(l, (\sigma, N_c))).$$

   When an event $\sigma_e \in \Sigma_e$ occurs in NS, it is sent through the control channel. This is represented by adding (σ, N_c) to l where N_c is the remaining time for σ_e until being received by G.

2. If $\delta_G(a, \sigma)!$ and $head(l) = (\sigma, 0)$:

   $$\delta_{NSP}((a, y, m, l), \sigma) = (\delta_G(a, \sigma), y, m \uplus [(\sigma, N_o)], tail(l)).$$

   An active controllable event $\sigma \in \Sigma_c$ can occur if the plant enables it, and the corresponding control command (enabling event) is received by the plant as $(\sigma, 0)$ (as the enabling event finished its journey through the control channel). When σ occurs, it will be stored in m with the remaining time N_o until being observed by NS.

3. If $\delta_G(a, \sigma)!$:

   $$\delta_{NSP}((a, y, m, l), \sigma) = (\delta_G(a, \sigma), y, m \uplus [(\sigma, N_o)], l).$$

   An uncontrollable event $\sigma \in \Sigma_{uc}$ can occur if it is enabled in G. When σ occurs, it will be stored in m with the remaining time N_o until being observed by NS.

4. If $\delta_G(a, tick)!$, $\delta_{NS}(y, tick)!$, $(\sigma, 0) \notin m$ for all $\sigma \in \Sigma_a$

   $$\delta_{NSP}((a, y, m, l), tick) = (\delta_G(a, tick), \delta_{NS}(y, tick), m - 1, l - 1).$$

   Event $tick$ can occur if both NS and G enable it, and there is no event ready to be observed by NS. Upon the execution of $tick$, all the time counters in m and l are decreased by one.

5. If $\delta_{NS}(y, \sigma_o)!$ and $(\sigma, 0) \in m$:

   $$\delta_{NSP}((a, y, m, l), \sigma_o) = (a, \delta_{NS}(y, \sigma_o), m \setminus [(\sigma, 0)], l).$$

   The observation of an active event $\sigma \in \Sigma_a$ can occur when it finishes its journey through the observation channel (and so it is received by NS), and σ_o is enabled by NS. When σ_o occurs, $(\sigma, 0)$ is removed from m. ■

In the rest of the chapter, the timed asynchronous composition $NS_{N_c}\|_{N_o} G$ of the plant G and the networked supervisor NS (for that plant) is assumed to be the TDES NSP represented by the automaton $(Z, \Sigma_{NSP}, \delta_{NSP}, z_0, Z_m)$.

Note that the networked supervised plant models the behavior of a plant controlled by a networked supervisor, and so for the proposed operator, we need to prove that the result does not enlarge the behavior of the plant.

Property 3.1 (NSP and Plant)**.** Given a plant G and networked supervisor NS (for that plant): $P_{\Sigma_G}(L(NSP)) \subseteq L(G)$.

Proof. See Appendix B.2.1. ■

A networked supervisor is controllable with respect to a plant if it never disables any uncontrollable event that can be executed by the plant. To have a formal representation of controllability in the NSC framework, Definition 3.2 is adapted to *timed networked controllability*.

Definition 3.8 (Timed Networked Controllability)**.** Given a plant G with uncontrollable events Σ_{uc} and forcible events Σ_{for}, a networked supervisor NS, is controllable w.r.t. G if for all $w \in L(NSP)$ and $\sigma \in \Sigma_{uc} \cup \{tick\}$, whenever $P_{\Sigma_G}(w)\sigma \in L(G)$:

1. $w\sigma \in L(NSP)$, or
2. $\sigma = tick$ and $w\sigma_f \in L(NSP)$ for some $\sigma_f \in \hat{\Sigma}_{for} \cup \Sigma_o$, where $\hat{\Sigma}_{for} = \Sigma_{for} \cup \Sigma_e$.

■

When there is no network, i.e., $\Sigma_{NS} = \Sigma_G$, timed networked controllability coincides with conventional controllability for TDES (Definition 3.2).

Remark. Considering Definition 3.7, *tick* does not occur if there is an event ready to be observed ($(\sigma, 0) \in m$). In other words, observed events always preempt *tick* since they occur once they finish their journey in the observation channel. The enabling events are assumed to be forcible as well. This gives the opportunity to the networked supervisor to preempt *tick* by enabling an event whenever it is necessary. In Section 3.3.3, we discuss other possible cases.

A networked supervisor NS is called proper in NSC framework if NS is timed networked controllable, and NSP is nonblocking and TLF. Similar to controllability, the definition of maximal permissiveness (in the conventional setting) is adapted to *timed networked maximal permissiveness* (for NSC Framework).

Definition 3.9 (Timed Networked Maximal Permissiveness)**.** A proper networked supervisor NS is timed networked maximally permissive for a plant G, if for any other proper networked supervisor NS' in the same NSC framework (with event set Σ_{NS}): $P_{\Sigma_G}(L({NS'}_{N_c}\|_{N_o} G)) \subseteq P_{\Sigma_G}(L(NSP)$. In other words, NS preserves the largest admissible behavior of G. ■

Again, when there is no network, this notion coincides with conventional maximal permissiveness (Definition 3.3).

Problem Statement: The *Basic NSC Problem* is defined as follows. Given a plant model G as a TDES, observation (control) channel with delay N_o (N_c) and maximum

capacity M_{max} (L_{max}), provide a networked supervisor *NS* such that

- *NSP* is nonblocking,
- *NSP* is time-lock free
- *NS* is timed networked controllable for G, and
- *NS* is timed networked maximally permissive.

3.3 Networked Supervisory Control Synthesis

To achieve a proper and maximally permissive networked supervisor (in the NSC framework), the synthesis is applied on the *networked plant*, as indicated in Figure 3.5. The networked plant is a model for how events are executed in the plant according to the enabling events, and how the observations of the executed events may occur in a networked supervisory control setting. Based on the networked plant, a synthesis algorithm is proposed to obtain a networked supervisor, which is a solution to the basic NSC problem. Example 3.4 is used to illustrate each step of the approach.

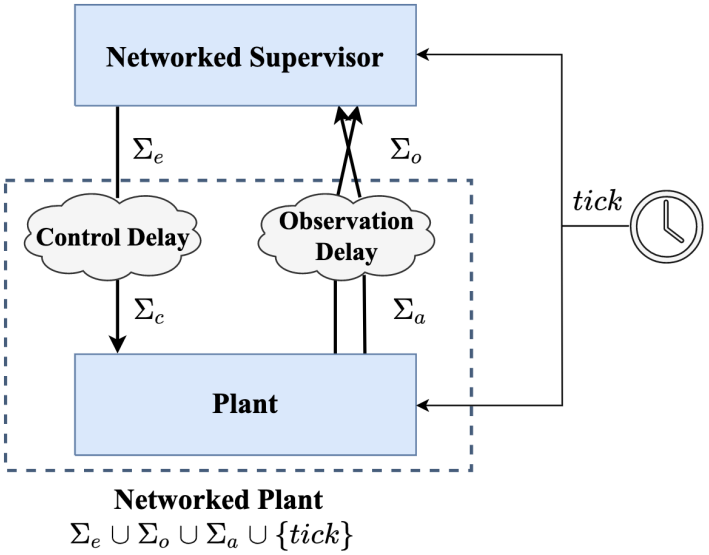


Figure 3.5: Networked plant.

Example 3.4. (Endangered Pedestrian) Let us consider the endangered pedestrian example from [Wonham 2015]. The plant G is depicted in Figure 3.6. Both the bus and pedestrian are supposed to do single transitions denoted by p for passing and j for jumping. The requirement considered in Wonham [2015] is that the pedestrian should jump before the bus passes. However, since we do not consider requirements here (yet), we adapt the plant from [Wonham 2015] such that if the bus passes before the pedestrian jumps, then G goes to a blocking state. The control channel is FIFO, the observation channel is non-FIFO, $N_c = N_o = 1$, $L_{max} = 1$, and $M_{max} = 2$. We aim to synthesize a proper and maximally permissive networked supervisor for G.

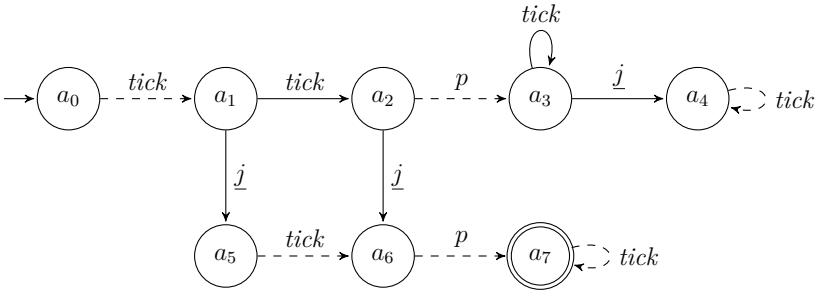


Figure 3.6: Endangered pedestrian from Example 3.4.

3.3.1 Networked Plant

The behavior of the plant communicating through the control and observation channels is captured by the networked plant. As is clear from Figure 3.5, if we do not consider enabling and observation of events, what is executed in the networked plant is always a part of the plant behavior. Let us denote by NP the networked plant automaton, then $P_{\Sigma_G}(L(NP)) \subseteq L(G)$.

Moreover, note that a networked supervisor is synthesized for a plant on the basis of the networked plant. The networked plant should represent all the possible behavior of the plant in the networked supervisory control setting, and it is only the networked supervisor that may prevent the occurrence of some plant events by disabling the relevant enabling event. This means that NP should be such that $L(G) \subseteq P_{\Sigma_G}(L(NP))$. The latter property relies on the following assumptions.

Assumption 1: The plant enables enough *ticks* in the beginning; there are at least N_c *ticks* (there can be uncontrollable events occurring between *ticks*) enabled before the first controllable event.

Assumption 2: The control channel provides enough capacity for all enabling commands being sent to the plant. Imagine that $tick\,\sigma\,tick^* \in L(G)$, and $L_{max} = 0$. Then, σ_e may occur in NP, but the plant will never execute σ as it does not receive the relevant enabling command. To avoid this situation, the size of the control channel should be such that it always has the capacity for all enabling events. An enabling event will be removed from the control channel after N_c *ticks*. So, considering all substrings w that can appear in the plant (after an initial part, say $w_0 \in \Sigma_G^*$), which are no longer (in the time sense) than N_c *ticks*, then the control channel capacity should be at least equal to the number of controllable events occurring in w; $L_{max} \geq \max_{w \in W} |P_{\Sigma_c}(w)|$ where $W = \{w \in \Sigma_G^* \mid \exists w_0 w \in L(G), |P_{\{tick\}}(w)| \leq N_c - 1\}$.

To obtain the networked plant, we present the function Π in Definition 3.10. In order to determine enabling commands we look N_c *ticks* ahead for only the controllable active events enabled in $G' = P_{\Sigma_G \setminus \Sigma_{uc}}(G)$. We use a list L to store the controllable events that have been commanded and a medium M to store the events that were executed.

Definition 3.10 (Networked Plant Operator)**.** For a given plant, G, Π gives the networked plant as:

$$\Pi(G, N_c, N_o, L_{max}, M_{max}) = (X, \Sigma_{NSP}, \delta_{NP}, x_0, X_m),$$

Let $G' = P_{\Sigma_G \setminus \Sigma_{uc}}(G) = (A', \Sigma_G, \delta'_G, a'_0, A'_m)$, and

$$\begin{aligned} X &= A \times A' \times M \times L, \\ x_0 &= (a_0, \delta'_G(a'_0, tick^{N_c}), [], \varepsilon), \\ X_m &= A_m \times A' \times M \times L. \end{aligned}$$

For $a \in A$, $a' \in A'$, $m \in M$ and $l \in L$, the transition function $\delta_{NP} : X \times \Sigma_{NSP} \to X$ is defined as follows:

1. If $\delta'_G(a', \sigma)!$, $\sigma \in \Sigma_c$

$$\delta_{NP}((a, a', m, l), \sigma_e) = (a, \delta'_G(a', \sigma), m, app(l, (\sigma, N_c))).$$

2. If $\delta_G(a, \sigma)!$, $head(l) = (\sigma, 0), \sigma \in \Sigma_c$

$$\delta_{NP}((a, a', m, l), \sigma) = (\delta_G(a, \sigma), a', m \uplus [(\sigma, N_o)], tail(l)).$$

3. If $\delta_G(a, \sigma)!, \sigma \in \Sigma_{uc}$

$$\delta_{NP}((a, a', m, l), \sigma) = (\delta_G(a, \sigma), a', m \uplus [(\sigma, N_o)], l).$$

4. If $\delta_G(a, tick)!$ and $(\sigma, 0) \notin m$ for all $\sigma \in \Sigma_a$

$$\delta_{NP}((a, a', m, l), tick) = \begin{cases} (\delta_G(a, tick), \delta'_G(a', tick), m - 1, l - 1) & \text{if } \delta'_G(a', tick)!, \\ (\delta_G(a, tick), a', m - 1, l - 1) & \text{otherwise.} \end{cases}$$

5. If $(\sigma, 0) \in m$

$$\delta_{NP}((a, a', m, l), \sigma_o) = (a, a', m \setminus [(\sigma, 0)], l).$$

■

Note that due to Assumption 1, $\delta'_G(a'_0, tick^{N_c})$ is always defined. In the rest of the chapter, the networked plant of the plant G is assumed to be the TDES NP represented by the automaton $(X, \Sigma_{NSP}, \delta_{NP}, x_0, X_m)$.

Property 3.2 (NP and Plant)**.** For any plant G:

1. $P_{\Sigma_G}(L(NP)) \subseteq L(G)$, and
2. $L(G) \subseteq P_{\Sigma_G}(L(NP))$ whenever assumptions 1 and 2 hold.

Proof. See Appendix B.2.2. ■

Example 3.5. For the endangered pedestrian from Example 3.4, G' and NP are given in Figure 3.7 and Figure 3.8, respectively.

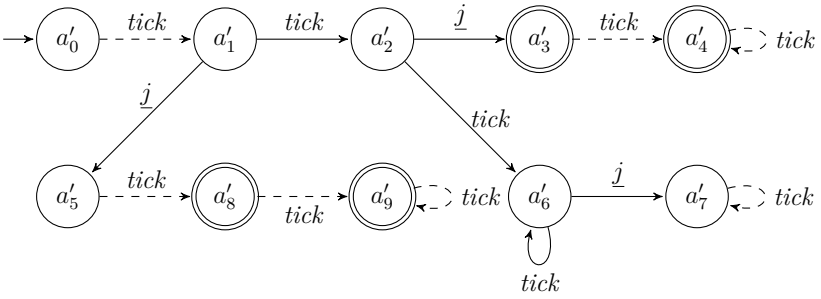


Figure 3.7: G' for the endangered pedestrian from Example 3.4.

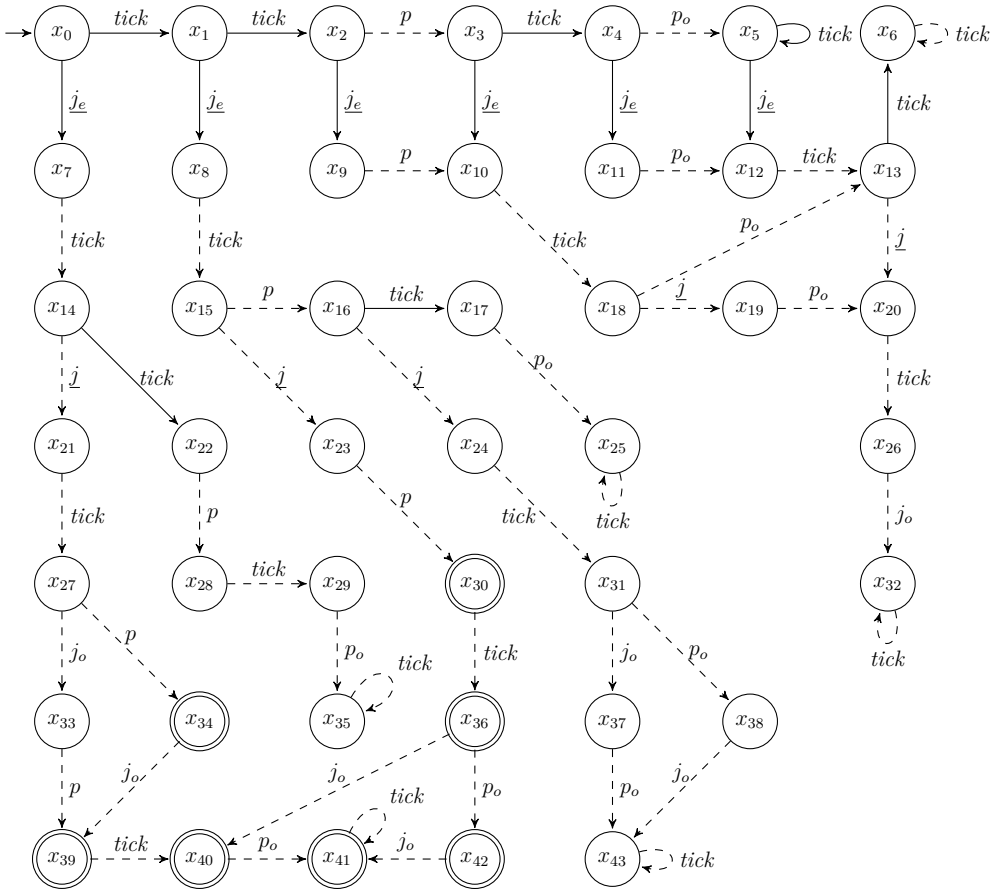


Figure 3.8: Networked plant for the endangered pedestrian from Example 3.4 ($N_c = N_o = 1$).

3.3.2 Synthesis

As is clear from Figure 3.5, enabling events are the only controllable events that can be disabled by the networked supervisor. All other events in the networked plant (active events and observed events) are uncontrollable. Moreover, controllability of *tick* depends on the forcible events of the plant as well as the enabling events (as we assume that they are forcible). To clarify, uncontrollable events are indicated by dashed lines in Figure 3.8. Note also that the observed events are observable to the networked supervisor. Also, events from Σ_e are observable, as the networked supervisor knows about the commands that it sends to

the plant. However, the events from Σ_a are now unobservable to the networked supervisor. To consider these issues in the current step of the approach, the sets of unobservable events $\hat{\Sigma}_{uo}$, observable events $\hat{\Sigma}_o$, uncontrollable active events $\hat{\Sigma}_{uc}$, and controllable active events $\hat{\Sigma}_c$ of the networked plant are given by $\hat{\Sigma}_{uo} = \Sigma_a$, $\hat{\Sigma}_o = \Sigma_e \cup \Sigma_o \cup \{tick\}$, $\hat{\Sigma}_{uc} = \Sigma_a \cup \Sigma_o$, $\hat{\Sigma}_c = \Sigma_e$. Also, note that forcing is an issue that is related to the modelling of the plant. A supervisor only decides whether forcing events is appropriate or not [Wonham and Cai n.d.]. Therefore, the forcing nature of the plant events is not changed in a networked supervisory control setting. So, $\hat{\Sigma}_{for} = \Sigma_{for} \cup \Sigma_e$. The event *tick* is always observable to the networked supervisor. Moreover, it is uncontrollable unless there exists an event from $\hat{\Sigma}_{for}$ enabled in parallel to *tick*. Regarding the new sets of events, the synthesis algorithm takes into account the TDES conventional controllability (in Definition 3.2) and is inspired from the weak observability condition introduced in Cai et al. [2016] and Takai and Ushio [2006].

Algorithm 3.1 presents the synthesis procedure in which we use the following additional concepts and abbreviations:

- $BS(NS) = BLock(NS) \cup TLock(NS)$ where $BLock(NS)$ gives the set of blocking states of NS, and $TLock(NS)$ gives the set of time-lock states of NS.
- Due to the fact that events from Σ_a are unobservable in the networked plant, one should be careful that the same control command is applied on the states reachable through the same observations. To take this issue into account, the following function is used in the synthesis algorithm; $OBS(x) = \{x' \in X \mid \exists w, w' \in \Sigma_{NP}^*, \delta_{NP}(x_0, w) = x \wedge \delta_{NP}(x_0, w') = x' \wedge P_{\hat{\Sigma}_o}(w) = P_{\hat{\Sigma}_o}(w')\}$ gives the set of states observationally equivalently reachable as x. The function OBS can be applied on a set of states $X' \subseteq X$ as well such that $OBS(X') = \bigcup_{x \in X'} OBS(x)$.
- $F(y) = \{\sigma \in \hat{\Sigma}_{for} \mid \delta_{NS}(y, \sigma)!\}$ is the set of forcible events enabled at state y.
- Besides blocking and time-lock states, we should take care of states from which a state from $BS(NS)$ can be reached in an uncontrollable way, taking preemption of *tick* events into account. $Uncon(BS(NS))$ is the smallest set of states, called *bad states*, such that
  1. $BS \subseteq Uncon(BS(NS))$;
  2. if $\delta_{NS}(y, \sigma) \in Uncon(BS(NS))$ for some $y \in Y$ and $\sigma \in \hat{\Sigma}_{uc}$, then $y \in Uncon(BS(NS))$;
  3. if $\delta_{NS}(y, tick) \in Uncon(BS(NS))$ for some $y \in Y$ such that for all $y' \in OBS(y)$, $F(y) \cap F(y') = \varnothing$, then $y \in Uncon(BS(NS))$. This is to make sure that the supervisor behaves the same towards all observationally equivalent transitions.
- $BPre(NS) = \{y \in Y \mid F(y) = 0 \wedge \neg\delta_{NS}(y, tick)! \wedge \delta_{NP}(y, tick)!\}$ contains states (still in NS) from which no forcible events and no *tick* are enabled while there was a *tick* event enabled in the networked plant.
- $Reach(NS)$ restricts an automaton to those states that are reachable from the initial state.

Algorithm 3.1 Networked supervisory control synthesis
Input: $NP = (X, \Sigma_{NSP}, \delta_{NP}, x_0, X_m)$, $\hat{\Sigma}_{uo}$, $\hat{\Sigma}_{uc}$, $\hat{\Sigma}_c$, $\hat{\Sigma}_{for}$
Output: $NS = (Y, \Sigma_{NS}, \delta_{NS}, y_0, Y_m)$

```
1:  i ← 0
2:  ns(0) ← NP
3:  bs(0) ← BS(ns(0))
4:  while y_0 ∉ Uncon(bs(i)) ∧ bs(i) ≠ ∅ do
5:      for y ∈ Y \ Uncon(bs(i)) and σ ∈ Σ̂_c ∪ {tick} do
6:          if δ_NS(y, σ) ∈ OBS(Uncon(bs(i))) then
7:              for y' ∈ OBS(y) do
8:                  δ_NS(y', σ) ← undefined
9:              end for
10:         end if
11:     end for
12:     Y ← Y \ Uncon(bs(i))
13:     i ← i + 1
14:     ns(i) ← Reach(ns(i − 1))
15:     bs(i) ← BPre(ns(i)) ∪ BS(ns(i))
16: end while
17: if y_0 ∈ Uncon(bs(i)) then
18:     no result
19: end if
20: NS ← P_{Σ_NSP \ Σ_a}(ns(i))
```

Starting from $NS = NP$, Algorithm 3.1 changes NS by disabling transitions at line 8 and delivering the reachable part at line 14. For the proposed algorithm, the following property and theorems hold.

Property 3.3 (Algorithm Termination)**.** The synthesis algorithm presented in Algorithm 3.1 terminates.

Proof. See Appendix B.2.3. ■

Theorem 3.1 (Nonblocking *NSP*)**.** Given a plant G and the networked supervisor NS computed by Algorithm 3.1: *NSP* is nonblocking.

Proof. See Appendix B.2.4. ■

Theorem 3.2 (TLF *NSP*)**.** Given a plant G and the networked supervisor NS computed by Algorithm 3.1: *NSP* is TLF.

Proof. See Appendix B.2.5. ■

Theorem 3.3 (Controllable *NS*)**.** Given a plant G and the networked supervisor NS computed by Algorithm 3.1: NS is timed networked controllable w.r.t. G.

Proof. See Appendix B.2.6. ■

Theorem 3.4 (Timed Networked Maximally Permissive *NS*)**.** For a plant G, the networked supervisor *NS* computed by Algorithm 3.1 is timed networked maximally permissive.

Proof. See Appendix B.2.7. ■

3.3.3 Possible Variants

The proposed synthesis approach can be adjusted for the following situations.

Nonblockingness or time-lock freeness

Algorithm 3.1 can easily be adapted to either only provide nonblockingness or time-lock freeness by removing $TLock(NS)$ and $BLock(NS)$ from $BS(NS)$, respectively.

Unobservable enabling events

We could have assumed that some events from Σ_e are unobservable. In this case, $\Sigma_a \subseteq \hat{\Sigma}_{uo} \subseteq \Sigma_a \cup \Sigma_e$, and so there would be more states that become observationally equivalent. Hence, the resulting supervisor could be more restrictive since a control command should be disabled at all observationally equivalent states if it needs to be disabled at one of them. Also if the observation channel does not provide enough capacity, more states become observationally equivalent, resulting in a more conservative solution. To not introduce any observation losses, the observation channel needs to be such that it has the capacity for all observations of events executed in the plant; $M_{max} \geq \max_{w \in W}\{|P_{\Sigma_a}(w)|\}$ where $W = \{w \in \Sigma_G^* \mid \exists w_0 w \in L(G), |P_{\{tick\}}(w)| \leq N_o\}$ as all events are observed after N_o *ticks*.

Non-forcible enabling events

We could have assumed that some events from Σ_e are not forcible. In this case, $\Sigma_{for} \subseteq \hat{\Sigma}_{for} \subseteq \Sigma_{for} \cup \Sigma_e$. Providing less forcible events makes the synthesis result more conservative since if the non-preemptable *tick* leads to a bad state, the current state where *tick* is enabled must be avoided as well (illustrated by Example 3.6).

Example 3.6. Consider the endangered pedestrian from Example 3.5. With the assumption that events from Σ_e are forcible, the networked supervisor is given in Figure 3.9. Without this assumption, there exists no networked supervisor.

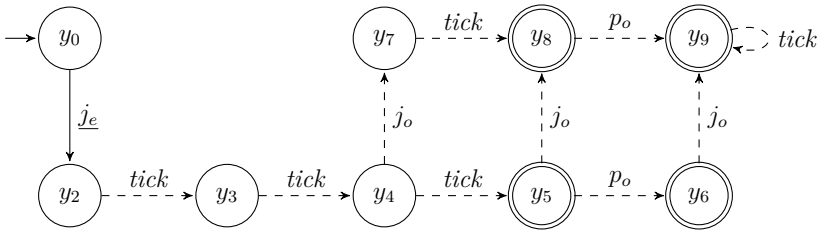


Figure 3.9: Networked supervisor for the endangered pedestrian from Example 3.4 ($N_c = N_o = 1$).

Non-FIFO control channel

Our proposed framework can easily be extended to the case that the control channel is non-FIFO by applying the following changes. Similar to the observation channel, the control channel is represented by $L = \{l \mid l : \Sigma \times [0, N_c] \to \mathbb{N}\}$ where l is a multiset. So, for each $l \in L$ and the time counter n, we define the operators $l \uplus [(\sigma, n)]$ and $l \setminus [(\sigma, 0)]$ instead of $app(l, (\sigma, n))$ and $tail(l)$, respectively. This affects item 1) of both Definition 3.7 and Definition 3.10 such that (σ, N_c) is simply added to l without taking into account the order of elements. Also, in item 2) of both definitions, $head(l)$ is replaced by $\exists(\sigma, 0) \in l$. This may change the result pretty much as the enabling events can now be received by G in any possible order. As Example 3.7 illustrates, this may increase the chance of reaching blocking or time-lock states and result in very conservative solutions for many applications.

Example 3.7. Given a plant G indicated in Figure 3.10, $N_c = N_o = 1$, and $L_{max} = M_{max} = 1$, NP is obtained as in Figure 3.11. The networked supervisor computed by Algorithm 3.1 only disables the event b_e at x_0. Now, assume that the control channel is non-FIFO as well. Then, at $x_3 = (\delta_G(a_0, tick), \delta'(a'_0, tick\, a\, b\, tick), [], (a, 0)(b, 0))$, b can be executed as well as a. By executing b at x_3, NP goes to a blocking state. In this case, Algorithm 3.1 returns no result since x_0 becomes a blocking state and needs to be removed. Note that this example aims to show how the synthesis result would be conservative if the control channel is FIFO. So, having no result conveys a positive message here.

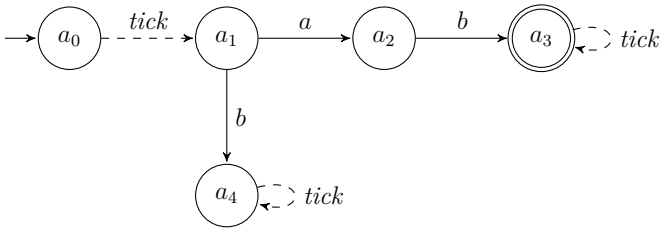


Figure 3.10: Plant from Example 3.7.

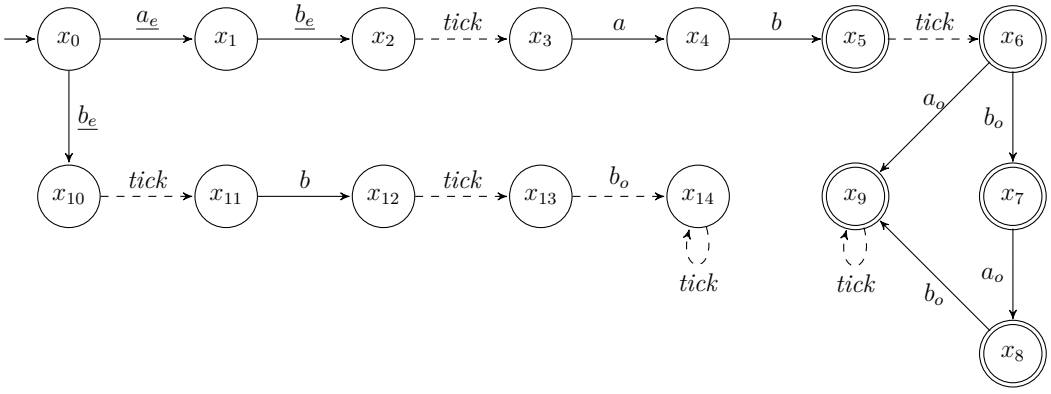


Figure 3.11: Networked plant from Example 3.7.

Delays specified to events

Our proposed approach can be adjusted to a setting with control delay N_c^σ and observation delay N_o^σ specified to each event $\sigma \in \Sigma_a$. The situation where events have different amounts of delays usually introduces similar effects as non-FIFO channels since the events may overtake each other. For instance, assume *tick tick a b tick tick* $\in L(G)$, where $N_o^a = 2$, $N_o^b = 1$, $N_c^a = 1$, and $N_c^b = 2$. Then, in the observation channel, b overtakes a, and b_o occurs before a_o. In the control channel, b needs to be enabled before a as b_e *tick* a_e *tick*, which results in $l = (b, 0)(a, 0)$. To have a networked pant satisfying Property 3.2, the execution of a should overtake the execution of b. Otherwise, the synthesis result would become very conservative. So, to adjust our approach, similar to the observation channel, the control channel is represented by a multiset m_c as well instead of a list. Then, the timed asynchronous composition operator (Definition 3.7) and the networked plant operator (Definition 3.10) need to be adjusted such that (σ, N_c^σ) is added to m_c instead of (σ, N_c) to l, and (σ, N_o^σ) is added to m instead of (σ, N_o). The synthesis algorithm does not need to change.

Bounded delays

Our proposed approach can be adjusted to a setting with non-constant control delay with lower bound (upper bound) as N_c^l (N_c^u) and observation delay with lower bound (upper bound) as N_o^l (N_o^u). In this situation, the timed asynchronous composition operator (Definition 3.7) and the networked plant operator (Definition 3.10) need to be adjusted such that $(\sigma, [N_c^l, N_c^u])$ is added to l instead of (σ, N_c), and $(\sigma, [N_o^l, N_o^u])$ is added to m instead of (σ, N_o). $l - 1$ and $m - 1$ reduce both the lower and upper bounds of events by one, and $l - 1$ removes an event with upper bound equal to zero from l. A controllable event, say σ, is executed if $head(l) = (\sigma, (0, i))$ for some $i \leq N_c^u$. Also, an event, say σ, is observed if $(\sigma, (0, i)) \in m$ for some $i \leq N_o^u$. The synthesis algorithm does not need to change.

3.4 Requirement Automata

To generalize the method to a wider group of applications, we solve the basic NSC problem for a given set of control requirements. It is assumed that the desired behavior of G, denoted by the TDES R, is represented by the automaton $(Q, \Sigma_R, \delta_R, q_0, Q_M)$ where $\Sigma_R \subseteq \Sigma_G$. Since most control requirements are defined to provide safety of a plant, we call a supervised plant *safe* if it satisfies the control requirements.

Definition 3.11 (Safety)**.** Given a plant G and requirement R, a TDES *NSP* with event set Σ_{NSP} is safe w.r.t. G and R if its behavior stays within the legal/safe behavior as specified by R; ; $P_{\Sigma_R}(L(NSP)) \subseteq L(R)$. ■

Problem Statement: Given a plant model G as a TDES, control requirement R for G (also a TDES), observation (control) channel with delay N_o (N_c) and maximum capacity M_{max} (L_{max}), provide a networked supervisor *NS* such that

- *NSP* is nonblocking,
- *NSP* is time-lock free,
- *NS* is timed networked controllable w.r.t. G,
- *NS* is timed networked maximally permissive, and
- *NSP* is safe for G w.r.t. R.

In the conventional non-networked supervisory control setting, if R is controllable w.r.t. G (as defined in Definition 3.2), then an optimal nonblocking supervisor can be synthesized for G satisfying R [Wonham 2015]. If R is not controllable w.r.t. G, then the supremal controllable sublanguage of $G||R$, denoted $sup\mathcal{C}(G||R)$, should be calculated. Then, the synthesis is applied on $sup\mathcal{C}(G||R)$ [Cassandras and Lafortune 2009; Wonham 2015].

In a networked supervisory control setting, synthesizing a networked supervisor for $sup\mathcal{C}(G||R)$ does not always result in a safe networked supervised plant. This issue occurs due to the fact that in $sup\mathcal{C}(G||R)$, some events are already supposed to be disabled, to deal with controllability problems introduced by requirement R. In a conventional non-networked setting, this does not cause a problem because events are observed immediately when executed. However, when observations are delayed, there could be a set of states reached by the same observation. Hence, if an event is disabled at a state, it should be disabled at all observationally equivalent ones. Even for a controllable requirement, any disablement of events should be considered at all observationally equivalent states. This issue is further clarified in Example 3.8.

To take care of this issue, any requirement automaton R (whether controllable or uncontrollable) is made complete as $R^{\perp}$ in terms of both uncontrollable and controllable events. *Completion* was first introduced in Flordal et al. [2007] where the requirement automaton R is made complete in terms of only uncontrollable events. By applying the synthesis on $G||R^{\perp}$, all original controllability problems in $G||R$ are translated to blocking issues. Note that this translation is necessary to let the supervisor know about the uncontrollable events that are disabled by a given requirement. To solve the blocking issues, synthesis still takes the controllability definition into account.

Definition 3.12 (Automata Completion)**.** For a TDES $R = (Q, \Sigma_R, \delta_R, q_0, Q_M)$, the complete automaton $R^{\perp}$ is defined as $R^{\perp} = (Q \cup \{q_d\}, \Sigma_R, \delta_R^{\perp}, q_0, Q_M)$ with $q_d \notin Q$, where for every $q \in Q$ and $\sigma \in \Sigma_R$,

$$\delta_R^{\perp}(q,\sigma) = \begin{cases} \delta_R(q,\sigma) & \text{if } \delta_R(q,\sigma)! \\ q_d & \text{otherwise.} \end{cases}$$

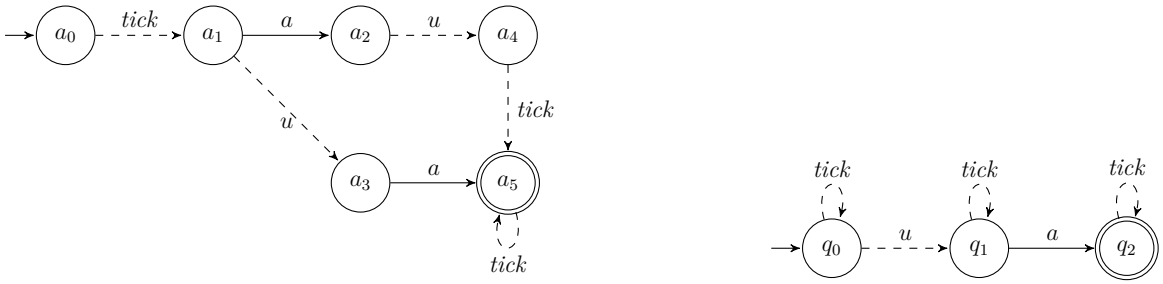
■

To find a networked supervisor, Algorithm 3.1 is applied on $\Pi(G||R^{\perp}, N_c, N_o, L_{max}, M_{max})$. The obtained networked supervisor is already guaranteed to be timed networked controllable, timed networked maximally permissive, and it results in a nonblocking and time-lock free networked supervised plant. Theorem 3.5 shows that the networked supervised plant is safe as well.

Theorem 3.5 (Safe *NSP*)**.** Given a plant G, requirement R, and the networked supervisor NS computed by Algorithm 3.1 for $\Pi(G||R^{\perp}, N_c, N_o, L_{max}, M_{max})$: $NS_{N_c}\|_{N_o}\ (G||R^{\perp})$ is safe for G w.r.t. R.

Proof. See Appendix B.2.8. ■

Example 3.8. Consider G depicted in Figure 3.12a for which $\Sigma_{uc} = \{u\}$ and $\Sigma_{for} = \varnothing$. The control objective is to design a supervisor satisfying the requirement that the event u must precede the event a, and not vice versa. This requirement can easily be modeled by the automaton shown in Figure 3.12b.



(a) Plant (b) Requirement

Figure 3.12: Plant and requirement for Example 3.8.

Let us select $N_o = N_c = L_{max} = M_{max} = 1$ and find a networked supervisor satisfying all the properties mentioned in the problem statement. For this purpose, $R^{\perp}$ is depicted in Figure 3.13. For NP^t depicted in Figure 3.14, Algorithm 3.1 does not result in a networked supervisor as the safe behavior of the plant cannot be preserved by any networked supervisor in the presence of delays. Note that if the requirement was made complete only in terms of uncontrollable events, Algorithm 3.1 would give the networked supervisor depicted in Figure 3.15, which results in the networked supervised plant depicted in Figure 3.16. The networked supervised plant is not safe as the event a may precede the event u.

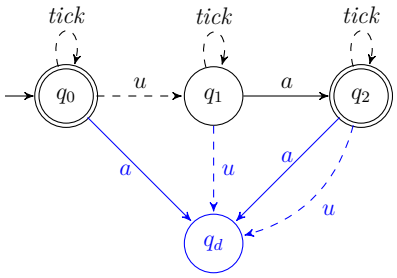


Figure 3.13: Total requirement for the plant stated in Example 3.8.

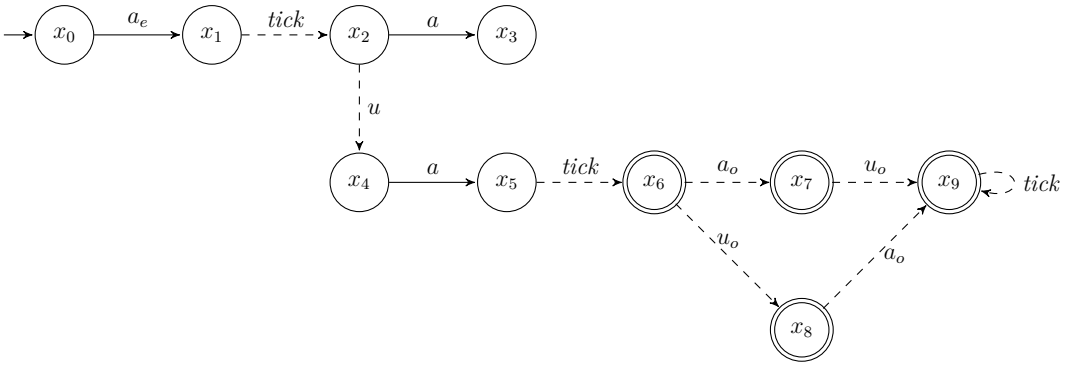


Figure 3.14: Networked plant from Example 3.8.

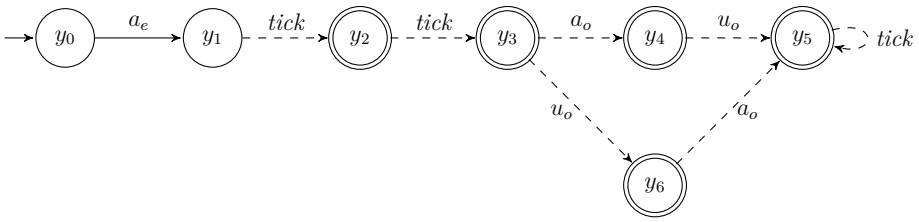


Figure 3.15: Unsafe networked supervisor for Example 3.8.

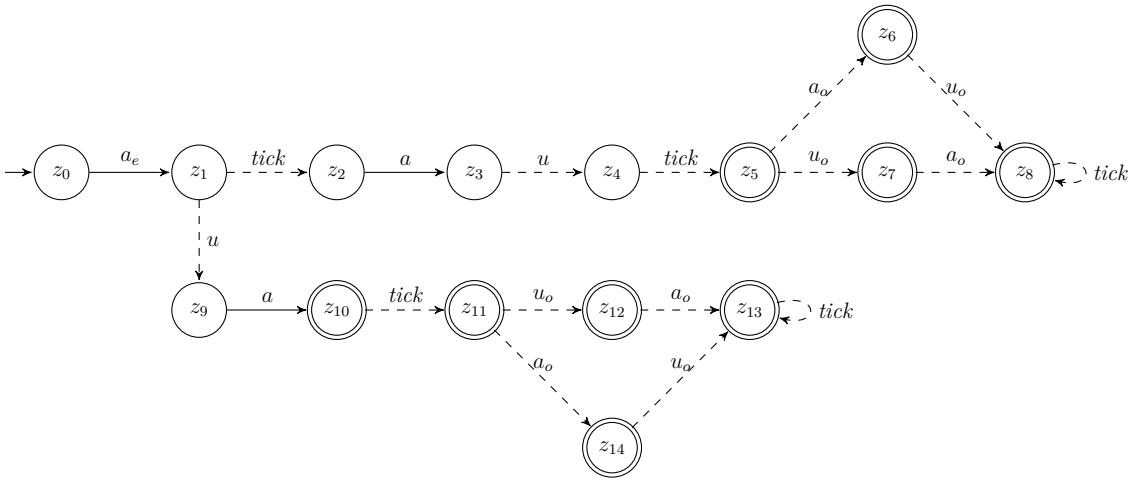


Figure 3.16: Unsafe networked supervised plant for Example 3.8.

3.5 Conclusions

In this chapter, we study the networked supervisory control synthesis problem. We first introduce a networked supervisory control framework in which both control and observation channels introduce delays, the control channel is FIFO, and the observation channel is non-FIFO. Moreover, we assume that a global clock exists in the system such that the passage of a unit of time is considered as an event *tick* in the plant model. Also, communication delays are measured as a number of occurrences of the *tick* event. In our framework, uncontrollable events occur in the plant spontaneously. However, controllable events can be executed only if they have been enabled by the networked supervisor. On the other hand, the plant can either accept a control command (enabled by the networked supervisor) and execute it or ignore the control command and execute some other uncontrollable event. For the proposed framework, we also provide a timed asynchronous composition operator to obtain the networked supervised plant. Furthermore, we adapt the definition of conventional controllability for our framework and introduce timed networked controllability. Then, we present a method of achieving the networked plant automaton representing the behavior of the plant in the networked supervisory control framework. For the networked plant, we provide an algorithm synthesizing a networked supervisor that is timed networked controllable, nonblocking, time-lock free, and maximally permissive. Finally, to generalize, we solve the problem for a given set of control (safety) requirements modeled as automata. We guarantee that the proposed technique achieves a networked supervisor that is timed networked controllable, nonblocking, time-lock free, maximally permissive, and safe.

For cases with large state spaces, we must deal with the scalability problem of the networked plant. For such cases, it is suggested to switch to timed automata.

Chapter 4

Supervisory Control of Timed Automata using Abstractions

Conventional supervisory control synthesis techniques are not adequate for timed automata (TA) due to the infinite state space. This chapter presents a supervisory control synthesis technique for TA with the objective of satisfying controllability and nonblockingness. The synthesis method consists of three steps. First, a TA is abstracted to a finite automaton (FA). The event set of the FA includes the discrete events of the TA as well as an event representing the passage of a considerable amount of time. Time passage is considered to be preemptable by events from a given set of forcible events. Second, an algorithm is presented to synthesize a controllable and nonblocking supervisor for the FA. Finally, a time-refinement technique is proposed to convert the supervisor to a TA.

4.1 Introduction

Discrete-event systems (DESs) are systems with a discrete set of states in which transitions occur only based on occurrences of asynchronous events [Cassandras and Lafortune 2009]. DESs are often modeled as finite automata (FA) [Hopcroft et al. 2006]. For the purpose of control, supervisory control theory of DESs has been developed by Ramadge & Wonham (RW-DES synthesis) [Ramadge and Wonham 1987]. RW-DES synthesis is a model-based theoretical framework for synthesizing a supervisor that restricts the behavior of a given plant towards a given set of control specifications (the desired behavior). Additionally, it takes care of controllability and nonblockingness [Cassandras and Lafortune 2009]. In

This chapter is based on Rashidinejad et al. [2020a]

DES, only the ordering of events matters, and time is totally neglected. However, it is not always sufficient to solely rely on the behavior of discrete events without taking into account the time at which these events occur.

Considering timing information in DESs is particularly important in a network-based supervisory control setting. In general, networked control of systems brings many advantages; it increases the flexibility of the system in terms of sharing data and adding/removing nodes, and it reduces the complexity and cost of the system by eliminating unnecessary wiring. However, networked control of systems brings challenges as well. In a networked control system, communication delays are unavoidable, and they have a high impact on the system behavior [Gupta and Chow 2010]. To consider the effects of communication delays, a precise model of timing is required. By considering time in DES, this chapter provides the basis for networked supervisory control approaches that are resilient to the effects of communication delays.

DESs in which time is also involved are referred to as real-time discrete-event systems (RTDESs) [Khoumsi 2002]. RTDESs are systems for which, in addition to the correct ordering of events, constraints on time delays must be satisfied [Khoumsi 2002]. To model RTDESs, two formalisms have been proposed: discrete-time models, and dense-time models. In discrete-time models, also known as timed DESs (TDESs), the discrete event *tick* is used to represent the passage of a unit of time of a global clock [Ostroff 1990]. To model a TDES, the typical DES model is augmented with time bounds for so-called active events [Dubey 2009]. A subset of the active events is considered to be forcible. A forcible event may preempt the passage of time. Therefore, the nature of the event *tick* lies between controllable and uncontrollable depending on the existence of an enabled forcible event. For the purpose of control, RW-DES synthesis has been extended for TDESs (RW-TDES synthesis) in Brandin and Wonham [1994]. A drawback of this approach is that each event has only a single lower and a single upper time bound, so that each occurrence of that event is only within its specified time bounds. Associating fixed time bounds to events limits the expressiveness of TDESs considerably.

Representing each tick of the clock by an event makes the model very sensitive to changes in the choice of the time unit. Consequently, it is difficult to model systems with different time scales, as the state space can quickly become very large. To consider dense-time in DES, Alur and Dill presented timed automata (TA) [Alur and Dill 1994]. TA consist of a finite set of locations that represent discrete states and a finite set of real-valued clocks to include the timing behavior [Dubey 2009]. The introduction of real-valued clocks makes TA more expressive than TDES [Ostroff 1990]. Compared to TDES, TA allow multiple and different time constraints on transitions. Additionally, TA provide a more compact graphical model, as time evolution is not shown explicitly.

The use of real-valued clocks in TA may result in an infinite state space, and therefore the existing synthesis approaches cannot be applied directly. To overcome this problem, a TA is first abstracted to an FA. Such abstractions have been proposed in Alur and Dill [1994]. To synthesize a supervisor for a TA, two main approaches have been presented in the literature: RW-based synthesis such as in Wong-Toi and Hoffmann [1991], and reactive synthesis (game-based synthesis) such as in Asarin et al. [1998], Cassez et al. [2005], Maler

et al. [1995], and Tripakis and Altisen [1999]. Game-based synthesis approaches for TA have been implemented in tools such as UPPAAL-TIGA [Behrmann et al. 2007; Maler et al. 1995]. The main difference between reactive and RW-based synthesis approaches is that reactive synthesis focuses on providing a possible winning strategy. However, RW-based synthesis provides a more permissive supervisor that includes *all* winning strategies. See [Ehlers et al. 2017] for a more detailed comparison of the approaches. Here, we are interested in RW-based synthesis approaches as we are looking for a supervisor that does not restrict the behavior of the plant more than necessary.

Supervisory control of TA has been firstly investigated in Wong-Toi and Hoffmann [1991] where the concepts of *untiming* and *timing* are introduced. Untiming converts a trace of a timed automaton into a region graph trace and timing converts a trace of a region graph into a set of TA traces. To synthesize a supervisor as a TA, the plant is first untimed to an FA called region graph using a region-based abstraction [Alur and Dill 1994; Wong-Toi and Hoffmann 1991]. For the untimed plant, a (untimed) supervisor is synthesized using the existing language-based methods for FA. Finally, the synthesized supervisor is timed. Region-based abstraction results in a finite but often very large region graph. However, it preserves enough information for synthesis [Tripakis and Altisen 1999].

An alternative abstraction approach was introduced in Khoumsi [2002], Khoumsi and Nourelfath [2002], and Ouedraogo et al. [2008, 2010] to transform a TA into a minimal and equivalent Set-Exp finite-state automaton (se-FSA). A new event *Set* is used to set or reset a clock timer, and an event *Exp* is used to indicate when this clock has expired. The synthesis procedure consists of combining the plant and specification into one TA, describing this TA as an se-FSA, and then computing a supervisor based on the RW-TDES synthesis procedure in Brandin and Wonham [1994]. Converting a TA to an se-FSA produces a smaller state space compared to the region-based abstraction approach. However, the resulting supervisor in this approach is an se-FSA, which requires an alternative control architecture. A method to transform an se-FSA back into a timed automaton does not exist [Ouedraogo et al. 2010].

Here, we focus on abstracting TA and applying the existing RW-based synthesis approach because it has been already implemented in toolsets such as in CIF [van Beek et al. 2014] or Supremica [Akesson et al. 2006]. Additionally, it is known that the RW-based synthesis provides a supervisor preserving the largest acceptable behavior of the plant. For this purpose, the plant is first abstracted to a region graph by applying the region-based abstraction technique from [Alur and Dill 1994; Wong-Toi and Hoffmann 1991]. Using the same idea from [Wong-Toi and Hoffmann 1991], the transition from a (clock) region to another one in the region graph is labeled by an event, called τ (indicating the passage of a considerable time delay). In this regard, we call the region graph a *τ-automaton*. Inspired from RW-TDES synthesis, the event τ is assumed to be preemptable by a set of forcible events. A synthesis algorithm is proposed to obtain a controllable and nonblocking (untimed) supervisor for the τ-automaton. The untimed supervisor (which is also a τ-automaton) is then refined to a timed automaton (timed supervisor) such that the disablement of events and time preemption (by the untimed supervisor) are translated to more restricted guards and invariants, respectively. By imposing additional timing restrictions on both guards and invariants, we synthesize a less

conservative (timed) supervisor compared to any other method that has been presented so far for (abstraction-based) supervisory control synthesis of TA that only allow restrictions on guards (see Section 4.5).

The contributions of this chapter are:

1. the synthesis technique we propose here is on automata level (instead of language level as in [Khoumsi 2002; Ouedraogo et al. 2010; Wong-Toi and Hoffmann 1991]) in order to be better suited for implementation.
2. we provide a time-refinement technique to transfer the (untimed) supervisor into a TA, which is not the case in Khoumsi [2002] and Ouedraogo et al. [2010].
3. we allow the (timed) supervisor not only to impose additional timing restrictions on guards but also on invariants, and so we provide a less conservative supervisor compared to [Khoumsi 2002; Ouedraogo et al. 2010; Wong-Toi and Hoffmann 1991].

The rest of the chapter is organized as follows. The definition of TA and the relevant concepts are given in Section 4.2. There, it is also discussed how a TA is abstracted to a τ-automaton. Then, the problem that is investigated in this chapter is formulated. In Section 4.3, a synthesis algorithm is proposed for a τ-automaton, which results in a controllable and nonblocking (untimed) supervisor in the form of a τ-automaton. In order to achieve the timed supervisor from the untimed supervisor and the original plant, a time-refinement technique is presented in Section 4.4. To clarify the synthesis procedure, an example is provided in Section 4.5. Finally, Section 4.6 concludes this chapter.

4.2 Background

Section 4.2.1 gives the definition of timed automata and the relevant concepts, many of which appear again in Chapter 5. Section 4.2.2 discusses the abstraction of TA to FA as introduced in Wong-Toi and Hoffmann [1991].

4.2.1 Timed Automata

A TA is an FA extended with a set of real-valued clocks.

Definition 4.1 (Timed Automaton [Alur and Dill 1994])**.** A timed automaton is a 7-tuple $(C, L, \Sigma, E, L_m, L_0, I)$ where

- C is a finite set of clocks with a non-negative real-value (from $\mathbb{R}_{\geq 0}$). The initial value of each clock variable is always assumed to be 0,
- L is a finite set of locations,
- Σ is a finite set of events,
- E is a finite set of edges with elements e of the form (l_s, σ, g, r, l_t) for which $l_s, l_t \in L$ are the source and target locations, respectively, $\sigma \in \Sigma$, g is the guard (clock constraint), which is a predicate over clock variables, and reset $r \subseteq C$,
- $L_m \subseteq L$ is the set of marked locations,

- $L_0 \subseteq L$ is the set of initial locations,
- I is a function associating an invariant to each location $l \in L$. An invariant is a clock constraint that needs to be satisfied when the system is in the location. ■

The guards and invariants are assumed to be represented by integer constraints over clock variables. Moreover, we frequently use the following notations:

- the notation . is used to refer to an element of a tuple. For instance, $e.\sigma$ refers to σ from $e \in E$.
- the notation $Pred[r]$ stands for a predicate $Pred$ and a reset r. The meaning of this notation is a predicate in which all occurrences of clock variables from r are replaced by zero. For instance, $(x > 1)[\{x\}]$ gives *false*.

Definition 4.2 (Clock Valuation)**.** Given a set of clocks C, a clock valuation $u : C \to \mathbb{R}_{\geq 0}$ assigns a non-negative real value to each clock $x \in C$. ■

For a clock valuation u, $u[r]$ assigns 0 to each clock $x \in r$. The values of $C \setminus r$ stay the same.

In this work, we only deal with deterministic TA.

Definition 4.3 (Deterministic TA [Alur and Dill 1994])**.** A timed automaton $(C, L, \Sigma, E, L_m, L_0, I)$ is deterministic if it has only one initial location, and for any pair of edges $e_1, e_2 \in E$, starting from the same location ($e_1.l_s = e_2.l_s$) and labeled by the same event ($e_1.\sigma = e_2.\sigma$), the clock constraints are mutually exclusive ($e_1.g \wedge e_2.g = \mathit{false}$). ■

Moreover, in examples, TA are depicted graphically. The locations are represented by circles and the edges by arrows from the source location to the target location, labelled with the event, the guard and the reset. The reset of a clock $c \in r$ is denoted $c := 0$. Invariants of locations are indicated inside the locations. Absence of an invariant in a location represents the invariant that always holds. The initial location is depicted by a dangling incoming arrow, and the marked locations by double circles.

Applications are typically modeled by a network of automata, where each automaton represents a single component or subsystem. A single automaton representing the network of automata can then be achieved as the synchronous product of the constituent automata [Alur and Dill 1994]. Synchronous product of TA is defined under the assumption that the two TA do not share any clock variable [Alur and Dill 1994; Bengtsson and Yi 2004]. This assumption is relaxed here, and the synchronous product is generalized for TA that may share clocks. To do so, we are inspired from the synchronous product of EFA from Skoldstam et al. [2007].

Definition 4.4 (Synchronous product of TA)**.** The synchronous product of two TA $G_1 = (C_1, L_1, \Sigma_1, E_1, L_{1m}, l_{10}, I_1)$ and $G_2 = (C_2, L_2, \Sigma_2, E_2, L_{2m}, l_{20}, I_2)$, is given by $G_1||G_2 = (C_1 \cup C_2, L_1 \times L_2, \Sigma_1 \cup \Sigma_2, E_p, L_{1m} \times L_{2m}, (l_{10}, l_{20}), I_p)$, where for each $l_1 \in L_1$ and $l_2 \in L_2$, $I_p(l_1, l_2) = I_1(l_1) \wedge I_2(l_2)$ and E_p is the smallest set that satisfies the following:

- whenever $\sigma \in \Sigma_1 \setminus \Sigma_2$, then for every $(l_{s1}, \sigma, g_1, r_1, l_{t1}) \in E_1$ and $l_2 \in L_2$, $((l_{s1}, l_2), \sigma, g_1, r_1, (l_{t1}, l_2)) \in E_p$

- whenever $\sigma \in \Sigma_2 \setminus \Sigma_1$, then for every $(l_{s2}, \sigma, g_2, r_2, l_{t2}) \in E_2$ and $l_1 \in L_1$, $((l_1, l_{s2}), \sigma, g_2, r_2, (l_1, l_{t2})) \in E_p$.
- whenever $\sigma \in \Sigma_1 \cap \Sigma_2$, then for every $(l_{s1}, \sigma, g_1, r_1, l_{t1}) \in E_1$ and $(l_{s2}, \sigma, g_2, r_2, l_{t2}) \in E_2$, $((l_{s1}, l_{s2}), \sigma, g_1 \wedge g_2, r_1 \cup r_2, (l_{t1}, l_{t2})) \in E_p$. ■

Every TA has an underlying semantic graph [Alur and Dill 1994; Tripakis and Yovine 2001].

Definition 4.5 (Semantic Graph)**.** The semantic graph of a TA $G = (C, L, \Sigma, E, L_m, l_0, I)$ is a labeled graph with a set of states $L \times (C \to \mathbb{R}_{\geq 0})$ consisting of a location and a clock valuation. The initial state is $(l_0, \mathbf{0})$, with $\mathbf{0}$ denoting the clock valuation where all the clocks are 0. The semantic graph has the following transitions:

- time transition: from state (l, u) to state $(l, u + \Delta)$ labeled with delay $\Delta \in \mathbb{R}_{\geq 0}$ if $u + \delta$ satisfies $I(l)$ for any δ such that $0 \leq \delta \leq \Delta$. Note that for a valuation u and a real value δ, $u + \delta$ denotes the clock valuation with $(u + \delta)(c) = u(c) + \delta$ for each clock c in the domain of u.
- event transition: from state (l_s, u_s) to state $(l_t, u_s[r])$ labeled by event σ if there is an edge $e = (l_s, \sigma, g, r, l_t)$ such that u_s satisfies g, and $u_s[r]$ satisfies $I(l_t)$.

Moreover, states (l, u) in the semantic graph with $l \in L_m$ (regardless of the clock valuation u) are marked. A word in the semantic graph of G is a finite sequence of labels (time or event). A state in the semantic graph of G is called reachable if it can be reached from the initial state via a word $w \in (\Sigma \cup \mathbb{R}_{\geq 0})^*$. The language of G, denoted $L(G)$, is the set of all words in its semantic graph starting from the initial state. ■

Based on the semantic graph, some relevant notions for timed automata are defined.

Definition 4.6 (Nonblockingness)**.** A state in a semantic graph is nonblocking if there exists a word leading from that state to a marked state, i.e., a state (l_t, u_t) with $l_t \in L_m$. A TA is nonblocking if all reachable states in its semantic graph are nonblocking. ■

The set of events of a TA is assumed to be partitioned into a set of uncontrollable events Σ_{uc} and a set of controllable events $\Sigma_c = \Sigma \setminus \Sigma_{uc}$. Uncontrollable events are events that occur spontaneously in the plant such as disturbances or sensor readings. Controllable events are signals sent to the actuators. In figures of TA edges labelled by uncontrollable events are indicated by dashed lines, and edges labelled by controllable events are indicated by solid lines. Time passage is uncontrollable by nature. However, it may be preempted by execution of a forcible event from a set of forcible events $\Sigma_{for} \subseteq \Sigma$ (forcible events are underlined in figures). Consequently, considering the semantic graph of a TA, a time transition enabled at a state is considered uncontrollable by default, unless there is also a forcible event transition enabled at that state. Then, the time transition is called *preemptable*. Note that a forcible event can be controllable or uncontrollable as discussed in Wonham [2015].

Definition 4.7 (Controllability of TA with forcible events)**.** Given a plant G with uncontrollable events Σ_{uc}, and forcible events Σ_{for}, a supervisor S is *controllable* w.r.t. G if for all $s \in L(S||G)$ and $\sigma \in \Sigma_{uc} \cup \mathbb{R}_{\geq 0}$, whenever $s\sigma \in L(G)$:

1. $s\sigma \in L(S||G)$, or
2. $\sigma \in \mathbb{R}_{\geq 0}$ and $s\sigma' \in L(S||G)$ for some $\sigma' \in \Sigma_{for}$.

Property (1) is the standard controllability property (when $\Sigma_{for} = \varnothing$, the time transition is always uncontrollable); S cannot disable uncontrollable events that may occur in G. However, if a forcible event is enabled, the time transition is preemptable, which is captured by Property (2). ■

4.2.2 Time-Abstraction

As previously discussed, to apply the RW-based synthesis approach on a TA, we first need to abstract the infinite state space of that TA into a finite one. In order to abstract a TA to an FA, while preserving information required for synthesis, the concept of region-based abstraction is used from [Alur and Dill 1994; Wong-Toi and Hoffmann 1991]. To achieve a finite representation for a TA G, its clock valuations are divided into a finite set of regions (denoted R_G) using the definition of region equivalence given in Alur and Dill [1994].

Definition 4.8 (Clock regions of TA [Alur and Dill 1994])**.** Consider a TA G with a set of clocks C where for each clock $x \in C$, c_x is the largest integer c such that $c \geq x$ or $x \leq c$ is a subformula of some clock constraint appearing in guards of edges and/or location invariants. Each clock region $R \in R_G$ is specified by:

1. for each clock $x \in C$, a clock constraint from $\{x = c \mid c = 0, 1, \ldots, c_x\} \cup \{c - 1 < x < c \mid c = 1, 2, \ldots, c_x\} \cup \{x > c_x\}$,
2. for any pair of $x, y \in C$ such that $c - 1 < x < c$ and $d - 1 < y < d$ appear in item 1) for some c and d, a clock constraint of one of the following forms: $x - c = y - d$, $x - c < y - d$, or $x - c > y - d$. ■

The abstracted FA in Alur and Dill [1994] contain only event transitions, and there is no time associated to these transitions. In Wong-Toi and Hoffmann [1991], the event τ is used to indicate the passage of some time causing the clock valuation to move from a clock region to the next one. Here, using the same idea, a TA is abstracted to a τ-automaton with respect to the set of clock regions R_G. The notation next(R) is used to denote the (uniquely defined) time successor of a region $R \in R_G$. Moreover, $R[r]$ denotes the clock region obtained by resetting all clock variables of r.

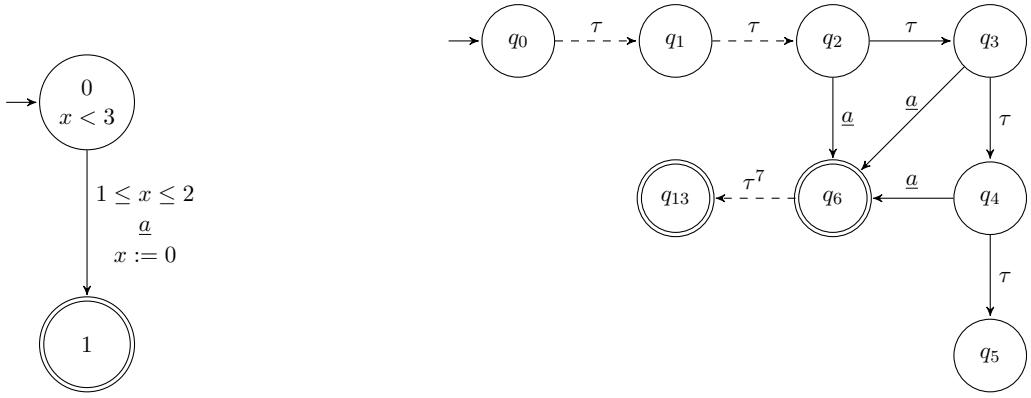
Definition 4.9 (τ-automaton of a TA)**.** Given a timed automaton $G = (C, L, \Sigma, E, L_m, l_0, I)$, the τ-automaton is an FA $P = (Q, \Sigma \cup \{\tau\}, \delta, q_0, Q_m)$ where $Q = L \times R_G$ is the set of states, $\Sigma \cup \{\tau\}$ is the set of events, $\delta : Q \times (\Sigma \cup \{\tau\}) \to Q$ is the (partial) transition function, $q_0 = (l_0, R_{\mathbf{0}})$ (where $R_{\mathbf{0}}$ is the region where all clocks are 0) is the initial state, and $Q_m = Q \cap (L_m \times R_G)$ is the set of marked states. The notation $\delta(q, \sigma)!$ denotes that δ is defined for state $q = (l, R)$ and event $\sigma \in \Sigma \cup \{\tau\}$ such that

- $\delta((l, R), \sigma) = (l', R[r])$ for any $(l, \sigma, g, r, l') \in E$ such that g is satisfied for the valuations represented by R.
- $\delta((l, R), \tau) = (l, \text{next}(R))$ such that $I(l)$ is satisfied by the valuations represented by R.

The sets of controllable, uncontrollable, and forcible events of P stay the same as of G. The event τ (showing the passage of time in the abstracted automaton) is generally an uncontrollable event. A τ-transition can be preempted by a forcible event, and if so it is called a *preemptable* τ-transition. ■

The following example clarifies the time-abstraction that is used here.

Example 4.1 (TA to τ-automata)**.** For the TA shown in the left in Figure 4.1, with clock x, and $\Sigma = \Sigma_c = \Sigma_{for} = \{a\}$, $R_G = \{x = 0, 0 < x < 1, x = 1, 1 < x < 2, x = 2, 2 < x < 3, x = 3, x > 3\}$. The τ-automaton achieved from the abstraction is depicted on the right. Each state of the τ-automaton indicates the current location of the TA, and the clock region that the current clock value belongs to (given in the table below the figures). Starting from q_0 only the event τ can occur as the guard of the edge labeled by a does not satisfy the clock region. This continues until the guard is satisfied for a valuation represented by the region, which is the case at states q_2, q_3, q_4. There, if the event a occurs, the location is changed from 0 to 1 and the clock is reset to 0 (at state q_6). Also, P can take a τ-transition as long as the invariant $I(0)$ is satisfied by the valuations represented by the clock region. Since a is forcible, the τ-transition enabled at q_2, q_3, q_4 is preemptable (indicated by solid lines). Also, we use the notation τ^n to refer to n consecutive occurrences of the event τ.



q_0	$(0, x = 0)$	q_2	$(0, x = 1)$	q_4	$(0, x = 2)$	q_6	$(1, x = 0)$
q_1	$(0, 0 < x < 1)$	q_3	$(0, 1 < x < 2)$	q_5	$(0, 2 < x < 3)$	q_{13}	$(1, x > 3)$

Figure 4.1: Abstraction of a TA to a τ-automaton.

Remark. In a deterministic TA, whenever there exist edges enabled from a location l that are labeled by the same event σ, the guards are mutually exclusive. Therefore, from each state in the τ-automaton corresponding to l, at most one of these guards is satisfied. As a result, the τ-automaton of a deterministic TA is also deterministic.

Here, we formalize the problem that we solve in this chapter.

Problem Statement: For a given TA G (representing a plant), the objective is to synthesize a TA S^t (a timed supervisor) that is controllable w.r.t. G, and the supervised plant $S||G$ is nonblocking.

The following sections discuss the synthesis procedure; in Section 4.3, an untimed supervisor S is synthesized for P. Then, in Section 4.4, the untimed supervisor S (a τ-automaton) is refined to a timed supervisor S^t (a TA).

4.3 Synthesis

This section presents a synthesis algorithm to achieve a non-blocking (untimed) supervisor for a τ-automaton P that results from the abstraction of a TA G.

The synthesis algorithm presented here is inspired from Chapter 3 and Rashidinejad et al. [2018], which present a state-based non-blocking supervisory control synthesis algorithm for a TDES under communication delays. Here, we present a similar synthesis algorithm in terms of time preemption for a τ-automaton with a set of forcible events.

Algorithm 4.1 results in an untimed supervisor $S = (Y, \Sigma \cup \{\tau\}, \delta_S, y_0, Y_m)$ (also as a τ-automaton) for P that avoids blocking states taking into account controllability. The algorithm starts from P and removes states that become unreachable through the algorithm (by disabling some transitions) so that the final result S has the set of states $Y \subseteq Q$.

In Algorithm 4.1, the following additional concepts are used:

- $BLock(S)$ gives the set of blocking states of S.
- $F_S(q) = \{\sigma \in \Sigma_{for} \mid \delta_S(q, \sigma)!\}$ is the set of forcible events enabled at state $q \in Y$.
- $Y_f(S) = \{q \in Y \mid \delta_S(q, \tau)! \wedge F_S(q) \neq \varnothing\}$ is the set of states of S at which the event τ is enabled, and it is preemptable.
- $Uncon_S(BS)$ gives the set of bad states, i.e., states from which a state from $BS \subseteq Y$ can be reached in S either through a sequence of uncontrollable events or uncontrollable (not preempt-able) τ's. $Uncon_S(BS)$ is the smallest set of states such that
  1. $BS \subseteq Uncon_S(BS)$;
  2. if $\delta_S(q, \sigma) \in Uncon_S(BS)$ for some $q \in Y$ and $\sigma \in \Sigma_{uc}$, then $q \in Uncon_S(BS)$;
  3. if $\delta_S(q, \tau) \in Uncon_S(BS)$ for some $q \in Y$ and $F_S(q) = \varnothing$, then $q \in Uncon_S(BS)$.
- $Reach(S)$ removes the unreachable states and the transitions to or from them.

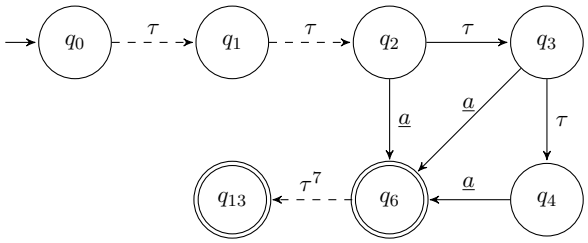
Example 4.2. Let us consider the τ-automaton from Example 4.1. The untimed supervisor synthesized from Algorithm 4.1 is depicted in Figure 4.2. To prevent the blocking state (q_5 of Figure 4.1), τ is preempted by the forcible event a at state q_4. The states of the untimed supervisor $Y \subseteq Q$ refer to the current location of the original plant, and the clock region that the current clock value belongs to (given in the table below the figure).

Remark. Consider the case where the event τ is enabled at a state, say $q \in Y$, where $F_S(q) \neq \varnothing$. Then, by definition $q \in Y_f(S)$. Now, imagine that τ is preempted at q by a forcible event, and in some later iterations, all those forcible events from $F_S(q)$ become disabled. Then, the τ-transition becomes uncontrollable, and so the state q is added to BS at line 12 as it does not belong to $Y_f(S)$ anymore. This issue is clarified in Example 4.3.

Algorithm 4.1 Untimed supervisory control synthesis

Input: $P = (Q, \Sigma \cup \{\tau\}, \delta, q_0, Q_m)$, Σ_{uc}, Σ_c, Σ_{for}
Output: $S = (Y, \Sigma \cup \{\tau\}, \delta_S, y_0, Y_m)$

1: $S \leftarrow P$
2: $Y_f \leftarrow Y_f(S)$
3: $BS \leftarrow BLock(S)$
4: **while** $y_0 \notin BS \wedge BS \neq \varnothing$ **do**
5: **for** $y \notin Uncon_S(BS)$ and $\sigma \in \Sigma_c \cup \{\tau\}$ **do**
6: **if** $\delta_S(y, \sigma) \in Uncon_S(BS)$ **then**
7: $\delta_S(y, \sigma) \leftarrow$ **undefined**
8: **end if**
9: **end for**
10: $Y \leftarrow Y \setminus Uncon_S(BS)$
11: $S \leftarrow Reach(S)$
12: $BS \leftarrow BLock(S) \cup (Y_f \setminus Y_f(S))$
13: **end while**
14: **if** $y_0 \in BS$ **then**
15: no result
16: **end if**



q_0	$(0, x = 0)$	q_2	$(0, x = 1)$	q_4	$(0, x = 2)$	q_{13}	$(1, x > 3)$
q_1	$(0, 0 < x < 1)$	q_3	$(0, 1 < x < 2)$	q_6	$(1, x = 0)$		

Figure 4.2: Untimed supervisor for τ-automaton from Example 4.1.

Example 4.3. Consider the τ-automaton shown in Figure 4.3. Here, the event b is forcible, and so the event τ enabled at state q_2 is controllable (represented by a solid line). Starting Algorithm 4.1, one gets $Y_f = \{q_2\}$ and $BS = \{q_3, q_4\}$. Consider the state q_2 from which $\delta_S(q_2, b) \in BS$ and $\delta_S(q_2, \tau) \in BS$ where $F_S(q_2) \neq \varnothing$. Then, due to line 7 of the algorithm, both the event b and τ become disabled, and $Y_f(S) = \varnothing$. At line 12, the state q_2 is considered a bad state to be removed in the next iterations when the event a becomes disabled. Finally, since q_0 becomes a bad state, the result will be an empty supervisor.

Remark. Starting Algorithm 4.1, $Y_f(S)$ and $BLock(S)$ both iterate in $\mathcal{O}(|Y|)$ steps by definition. Also, the while-loop iterates in $\mathcal{O}(|Y|)$ steps. Inside the while-loop, the for-loop iterates in $\mathcal{O}(|Y| \times (|\Sigma_c| + 1))$ since in the worst case, all the controllable events and the preemptable τ-transition are disabled at any state. Each of $Uncon_S(BS)$ and $Reach(S)$ also terminates in $\mathcal{O}(|Y|)$ steps by definition. So, the complexity of Algorithm 4.1 is $\mathcal{O}(|Y|^2(|\Sigma_c| + 1))$. However, in most cases, the algorithm terminates faster than the

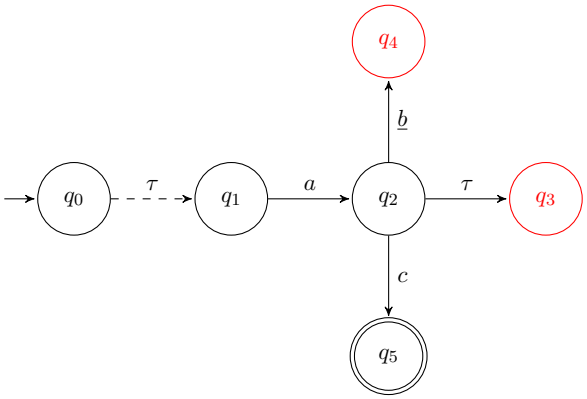


Figure 4.3: τ-automaton from Example 4.3.

worst case. For instance, for the bus-pedestrian in Section 4.5, although the worst case complexity is 288 iterations, the algorithm terminates after 2 iterations.

4.4 Time-Refinement

In this section, a time-refinement technique is presented to obtain the timed supervisor S^t based on the untimed supervisor S and the original plant G. The idea is to translate the decisions made by S for P to decisions that S^t should make for G. Applying Algorithm 4.1 to a τ-automaton P (as the abstracted plant) with a set of (symbolic) states Q results in S that has a set of states $Y \subseteq Q$. Therefore, each state $q \in Y$ is a pair of $l \in L$, and a clock region $R \in R_G$. This information is used to determine S^t where we introduce the following concepts; $K(l) = \{q \in Y \mid q = (l, R)\}$ is a set of states related to the same location l of the TA, and $J(l, \sigma) = \{q \in K(l) \mid \delta_S(q, \sigma)!\}$ is a subset of $K(l)$ from which the event σ is enabled. $K(l)$ is used to determine the invariant of the location l as it gives any pair of l and the possible (clock) region that l can be in. $J(l, \sigma)$ is used to determine the guard of each edge labeled by σ as it gives any pair of l and the clock region where the event σ can occur. Consider the untimed supervisor S given in Figure 4.2. Then, $K(0) = \{q_0, q_1, q_2, q_3, q_4\}$ is the set of states related to the same location $l = 0$ and $K(1) = \{q_6, q_{13}\}$ is related to location $l = 1$. $J(0, a) = \{q_2, q_3, q_4\}$ is a subset of $K(0)$ from which the event a is enabled. The transformation of S to S^t is as follows:

Definition 4.10. Given a plant $G = (C, L, \Sigma, E, L_m, l_0, I)$, the abstraction of G to the τ-automaton P, and the untimed supervisor $S = (Y, \Sigma \cup \{\tau\}, \delta_S, y_0, Y_m)$ obtained by applying Algorithm 4.1 on P, the timed supervisor is $S^t = (C, L, \Sigma, E_s, L_m, l_0, I_s)$, where E_s and I_s are determined as follows:

- $E_s = \{(l_s, \sigma, \bigvee_{q \in J(l_s,\sigma)} q.R, -, l_t) \mid \delta_S((l_s, R_s), \sigma) = (l_t, R_t)\}$.
- for each location $l \in L$, $I_s(l) = \bigvee_{q \in K(l)} q.R$. ■

Note that the (timed) supervisor does not influence the resets in the plant, and so the resets are left out in the edges (denoted by $-$). Considering Definition 4.10, if a state (l, R) in P becomes unreachable in S, it follows that $(l, R) \notin K(l)$, and so R will not be

considered in $I_s(l)$ (see Example 4.4). Also, in the case that the event σ is disabled by S at state $q = (l, R)$, then $q \notin J(l, \sigma)$. So, in the timed supervisor, R will not be considered in the guard of the edge labeled by σ (see Example 4.5).

Example 4.4. Consider the untimed supervisor S given in Figure 4.2. The timed supervisor is presented in Figure 4.4 for which, considering Definition 4.10, $E_s = \{(0, a, q_2.R \vee q_3.R \vee q_4.R, -, 1)\}$, $I_s(0) = q_0.R \vee q_1.R \vee q_3.R \vee q_4.R$, and $I_s(1) = q_6.R$ (which is always true). Note that the invariants and guards are simplified in the figure.

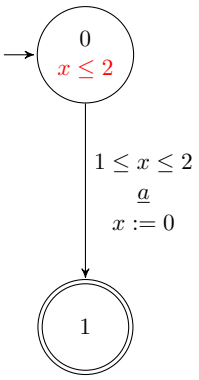
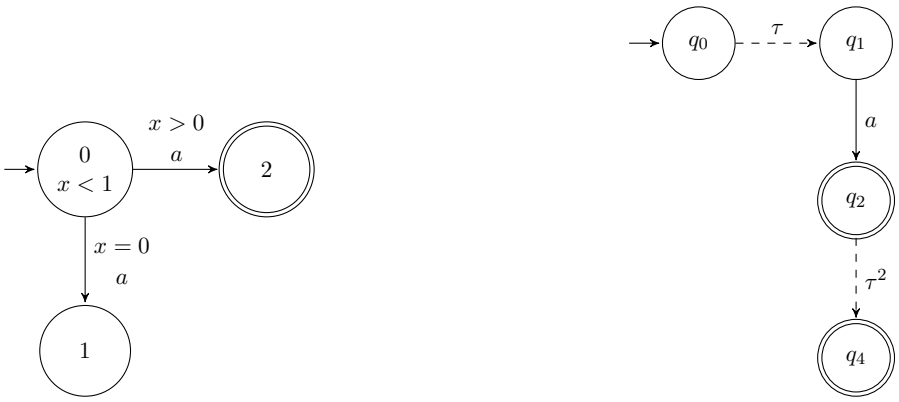


Figure 4.4: Timed supervisor for TA from Example 4.1.

Example 4.5. Consider the TA G and its untimed supervisor S given in Figure 4.5 (G in the left and S in the right). $K(0) = \{q_0, q_1\}$, $K(1) = \varnothing$, $K(2) = \{q_2, q_3\}$, and $J(0, a) = \{q_1\}$. From Definition 4.10, we achieve the timed supervisor S^t depicted in Figure 4.6. Note that location 1 is unreachable in S^t as the transition $\delta((0, x = 0), a) = (1, x \geq 0)$ in P is undefined in S and so $q_0 \notin J(0, a)$.



q_0	$(0, x = 0)$	q_2	$(2, 0 < x < 1)$
q_1	$(0, 0 < x < 1)$	q_4	$(2, x > 1)$

Figure 4.5: Timed automaton and its untimed supervisor from Example 4.5.

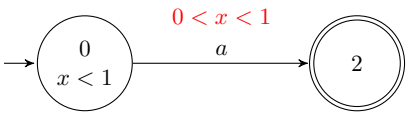


Figure 4.6: Timed supervisor for the TA from Figure 4.5.

Theorem 4.1 summarizes the main result of this chapter.

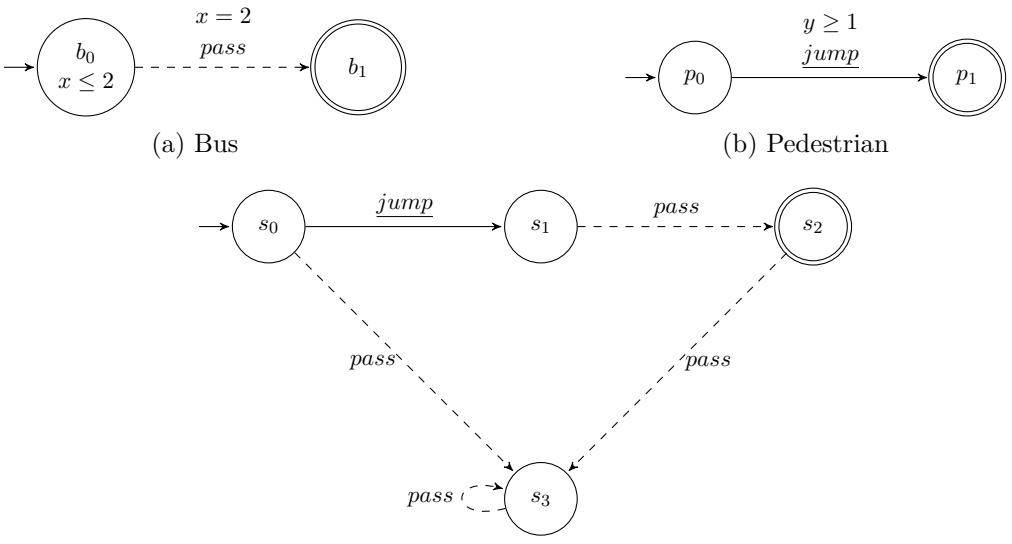
Theorem 4.1. Consider a TA G and its corresponding τ-automaton P for which the untimed supervisor S results from Algorithm 4.1. The timed supervisor S^t achieved from time-refinement of S is controllable for G, and $S^t||G$ is nonblocking.

Proof. See Appendix C.2.1. ■

4.5 Example: Bus-Pedestrian

To clarify the synthesis procedure, the bus-pedestrian example from Khoumsi and Nourelfath [2002] is used.

Consider a bus that is headed directly for a pedestrian and will run over him at $x = 2$ time units if he does not move. The pedestrian needs $y = 1$ time unit to realize his fate, after which he has the chance to jump out of the bus's path. If the pedestrian jumps before the bus passes, he is safe. Figure 4.7 gives the automata representing the bus, pedestrian, and the safe behavior of the system. The safe behavior is modeled in such a way that if the pedestrian jumps before the bus passes, then the system goes to a marked state. Otherwise, the system goes to a blocking state. The event *pass* is uncontrollable, and the event *jump* is forcible.



(a) Bus

(b) Pedestrian

(c) Safe behavior

Figure 4.7: Plant automata for the bus-pedestrian example.

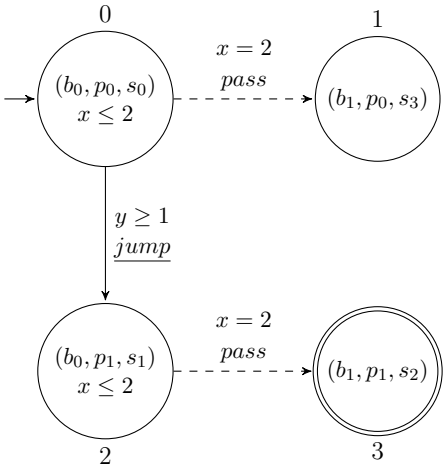
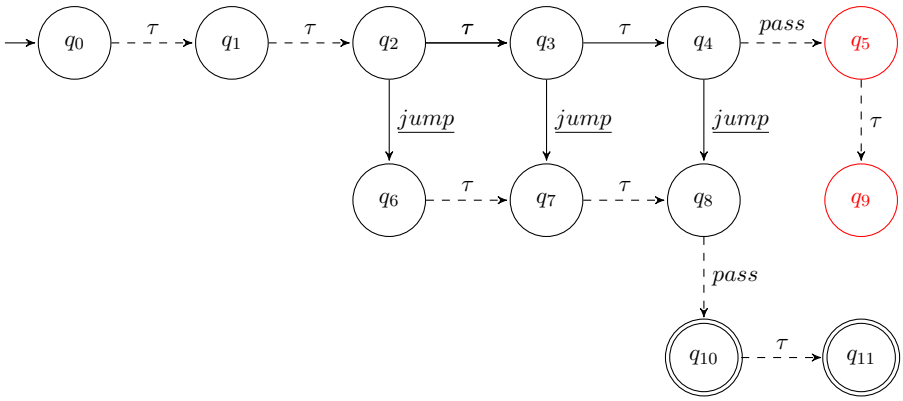


Figure 4.8: Synchronous product of the TA from Figure 4.7.

The synchronous product of the bus, pedestrian and the safe behavior automata is shown in Figure 4.8. To apply synthesis, the TA is first abstracted to the τ-automaton shown in Figure 4.9 where the blocking states are indicated in red.



q_0	$(0, x = y = 0)$	q_4	$(0, x = y = 2)$	q_8	$(2, x = y = 2)$
q_1	$(0, 0 < x = y < 1)$	q_5	$(1, x = y = 2)$	q_9	$(1, x = y > 2)$
q_2	$(0, x = y = 1)$	q_6	$(2, x = y = 1)$	q_{10}	$(3, x = y = 2)$
q_3	$(0, 1 < x = y < 2)$	q_7	$(2, 1 < x = y < 2)$	q_{11}	$(3, x = y > 2)$

Figure 4.9: Abstraction of the TA from Figure 4.8 to a τ-automaton.

By applying Algorithm 4.1 to the τ-automaton depicted in Figure 4.9, we get an untimed supervisor S shown in Figure 4.10. As it is clear from the figure, to take care of controllability and nonblockingness, S disables the event τ at state q_3.

The final step is to apply the time-refinement technique on S. Considering Figure 4.10, $K(0) = \{q_0, q_1, q_2, q_3\}$ and $J(0, jump) = \{q_2, q_3\}$. Considering Figure 4.8, the timed supervisor applies the following restrictions; 1) the guard of the edge labeled by *jump* is set to $x = y = 1 \vee 1 < x = y < 2$, which simplifies to $1 \leq x = y < 2$, and 2) the invariant of location 0 is set to $x = y = 0 \vee 0 < x = y < 1 \vee x = y = 1 \vee 1 < x = y < 2$, or

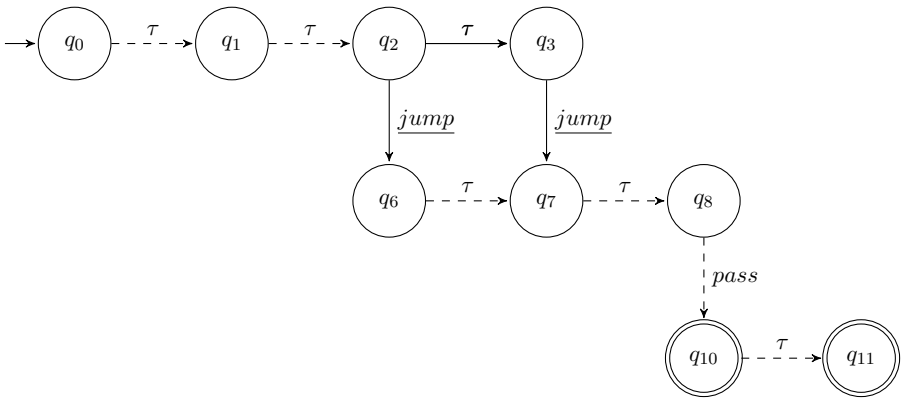


Figure 4.10: Untimed supervisor S for the TA from Figure 4.7.

equivalently $x = y < 2$. The resulting timed supervisor is given in Figure 4.11.

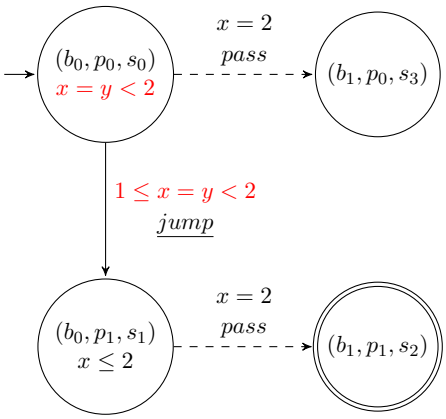


Figure 4.11: Timed supervisor S^t for the TA from Figure 4.7.

The supervised plant is $S^t||G = S^t$, which is obtained from Definition 4.4. The supervised plant leaves state (b_0, p_0, s_0) before the guard of the uncontrollable event $pass$ is satisfied, and so it never reaches the blocking state (b_1, p_0, s_3). Comparing the supervised plant and the (uncontrolled) plant, the invariant of location (b_0, p_0, s_0) is restricted due to the disablement of τ by S at state q_3 (in Figure 4.10). Moreover, the guard of the edge labeled by the controllable event $jump$ is restricted due to the event disablement by S at state q_4 of P, and also to match the invariant. Note that this does not restrict the behavior of the plant as time could not have passed the guard limit due to the invariant. The resulting (timed) supervisor prevents the nonblocking state of the TA while it takes care of controllability.

For many applications, the supervisor may need to preempt time by forcing the plant to take an action. For such cases, we need the concept of forcible events. Considering the bus-pedestrian example, if the event $jump$ was not forcible (the supervisor could not force the event $jump$ to preempt time), then the result would be an empty supervisor is the case with the approaches in Khoumsi [2002] and Wong-Toi and Hoffmann [1991], since additional timing restrictions on invariants are not allowed.

4.6 Conclusions

In this chapter, we present a supervisory control synthesis technique for timed automata (TA). Modeling plants as TA makes it possible to consider communication delays based on real-time. By investigating supervisory control of timed automata, this chapter builds the basis of networked supervisory control synthesis in a real-time setting. The objective is to synthesize a supervisor also as a TA, which guarantees controllability and nonblockingness. For this purpose, region-based abstraction is used to abstract a TA to a finite automaton, called τ-automaton, in which the event τ is used to indicate the passage of some time delay. The event τ is assumed to be uncontrollable; however, it can be preempted by a forcible event. A state-based synthesis algorithm is proposed for the τ-automaton, which results in a nonblocking and controllable (untimed) supervisor also as a τ-automaton. Finally, a time-refinement technique is presented to convert the untimed supervisor (as a τ-automaton) to a timed supervisor (as a TA).

Chapter 5

Supervisory Control of Timed Automata without Abstractions

Considering real-valued clocks in timed automata (TA) makes it a practical modeling framework for discrete-event systems. However, the infinite state space brings challenges to the control of TA. To synthesize a supervisor for TA using the conventional supervisory control theory, existing methods abstract TA to finite automata (FA). For many applications, the abstraction of real-time values results in an explosion in the state space of FA. This chapter presents a supervisory control synthesis algorithm directly applicable to the TA without any abstraction. The plant is given as a TA with a set of uncontrollable events and a set of forcible events. Forcible events can preempt the passage of time when needed. The synthesis algorithm works by iteratively strengthening the guards of edges labeled by controllable events and invariants of locations where the progression of time can be preempted by forcible events. The synthesized supervisor, which is also a TA, is guaranteed to be controllable, maximally permissive, and results in a nonblocking and safe supervised plant.

5.1 Introduction

Supervisory control theory (SCT) was first introduced by Ramadge-Wonham to control discrete-event systems (DESs) [Ramadge and Wonham 1987]. SCT provides a synthesis method resulting in a supervisor that restricts the plant behavior towards a given set

This chapter is based on Rashidinejad et al. [2021b] with an earlier version as Rashidinejad et al. [2020b].

of desired behavior. Moreover, the synthesized supervisor satisfies the controllability, nonblockingness, and maximal permissiveness properties [Wonham 2015].

DESs, such as communication networks, manufacturing and traffic systems, are typically modeled using finite automata (FA). To provide a compact representation of complex and large DESs, FA have been further extended with discrete variables to extended finite automata (EFA) [Skoldstam et al. 2007]. In EFA, transitions are labeled by events and associated with constraints on variables (guards), where variables may be updated after the occurrence of an event [Skoldstam et al. 2007].

The dynamics of DESs depend entirely on the ordering of the event occurrences, and so are independent of time [Cassandras and Lafortune 2009]. However, the control of many applications needs to be able to include timing information in modeling DESs. Imagine a system that needs to be controlled over a distance, due to being located in a hazardous or unreachable environment. To control such systems, the concept of networked supervisory control is introduced in Chapter 3.

Networked control of systems introduces communication delays that are unavoidable and have a high impact on the system performance [Heemels et al. 2010]. To consider the effects of communication delays, the DES model must include timing information of event occurrences as well as the ordering of them. For this purpose, the concepts of timed discrete-event systems (TDESs), and timed automata (TA) have been introduced in Brandin and Wonham [1994] and in Alur and Dill [1994], respectively. TDESs and TA are known as real-time discrete-event systems (RTDESs), which are modeled not only based on the ordering of events, but also based on timing constraints on events [Khoumsi 2002].

TDESs incorporate discrete time in modeling DESs. A TDES is generally a DES in which the execution of each event, called *active* event, is restricted within a lower and an upper time bound specified for the event. It is assumed that a digital clock exists in the system, and so the TDES is modeled as a FA that includes a specific event, called *tick*, indicating the passage of a unit of time. The event *tick* is generally an uncontrollable event as it spontaneously occurs in the system, and so it cannot be disabled by a supervisor. However, it is assumed that *tick* is *preemptable* by a subset of active events, called *forcible* events. Taking the nature of *tick* into account, SCT of DESs, has been modified for TDESs in Brandin and Wonham [1994]. Moreover, like DESs, the model of TDESs has been extended with discrete variables into timed extended finite automata (TEFA) [Miremadi et al. 2015].

TA incorporate dense-time in modeling DESs [Alur and Dill 1994]. A TA consists of a finite set of locations and a finite set of real-valued clocks [Dubey 2009]. To each location, a clock constraint is associated, called an *invariant*, determining the time that the system is allowed to stay in that location. Each edge between two locations is labeled by an event, the clock constraint associated to that event called the *guard*, and the set of clocks that are reset to zero, called the *reset*, by the occurrence of that event.

Compared to TDESs, a TA brings a more natural modeling framework for real-life applications because 1) it considers real-time, and so it copes with the state-space explosion problem introduced by discrete time; this is especially important for systems with various

time scales. And 2) it easily allows events to have multiple and different timing constraints, rather than specifying the time of each event occurrence by fixed lower and upper bounds.

The control of TA is challenging due to the clock variables, making the state space of TA infinite. To overcome this problem, existing approaches abstract TA into FA, and apply supervisory control synthesis on the abstracted result [Maler et al. 1995; Tripakis and Altisen 1999; Wong-Toi and Hoffmann 1991]. In general, the synthesis approaches can be divided into the following categories: 1) game-based (reactive) synthesis, and 2) the synthesis method proposed by Ramadge-Wonham, which is referred to as RW-based synthesis here. Game-based (reactive) synthesis of TA has been investigated in Asarin et al. [1998], Cassez et al. [2005], Maler et al. [1995], and Tripakis and Altisen [1999], and it has also been implemented in tools such as UPPAAL-TIGA [Behrmann et al. 2007; Maler et al. 1995]. Game-based synthesis and RW-based synthesis mainly differ in satisfying maximal permissiveness. Game-based synthesis gives a winning strategy if it exists. However, RW-based synthesis provides a unique maximally permissive supervisor, which compromises *all* winning strategies [Ehlers et al. 2017]. Here, we focus on RW-based synthesis as we want to achieve a maximally permissive, controllable, and nonblocking supervisor.

RW-based supervisor synthesis of TA was first investigated in Wong-Toi and Hoffmann [1991], where the plant is first abstracted into an FA (region graph) using region-based abstraction from [Alur and Dill 1994; Wong-Toi and Hoffmann 1991]. Then, a supervisor is synthesized for the FA using existing methods. Finally, to refine the abstraction, timing information is added to the FA supervisor. For many applications, region-based abstraction results in a finite but a very large FA [Khoumsi and Nourelfath 2002; Tripakis and Yovine 2001].

To overcome the state-space explosion problem of region-based abstraction, some state-space minimization methods have been proposed such as zone-based abstraction [Alur and Dill 1994]. These methods are mainly used for model checking and verification purposes as they do not provide sufficient information for supervisor synthesis [Ouedraogo et al. 2010].

In Khoumsi and Nourelfath [2002] and Ouedraogo et al. [2010], a transformation is introduced to obtain a minimal FA from a TA that is suitable for synthesis purposes. The transformation is based on two special events; *Set* and *Exp*, where *Set* represents the set and reset of a clock, and *Exp* indicates the expiration of the clock. The *SetExp*-transformation results in a minimal FA, for which a supervisor is synthesized using the concept of forcible events from TDESs. Preempting time using forcible events results in a more comprehensive solution as more events can be disabled if needed. However, it is currently unknown how to refine the synthesized supervisor (as an FA with *Set* and *Exp* events) to a TA (with these events translated into time constraints), and so the synthesis based on *SetExp*-transformation is not satisfying.

Supervisory control of TA using forcible events has already been studied in Chapter 4, where region-based abstraction is used to abstract a TA into an FA. For the FA, a synthesis algorithm is proposed. The synthesized supervisor is transformed back into a TA using a time-refinement technique. Although this method gives the supervisor as a TA, it still suffers from the state-space explosion problem caused by the abstraction.

Here, we provide a supervisory control technique for TA such that:

- no abstraction is needed to cope with the state-space explosion problem,
- an algorithm is proposed that works with automata instead of languages to ease integration of an implementation in a tool set such as CIF [van Beek et al. 2014] or Supremica [Akesson et al. 2006],
- the RW-based synthesis is used so that the synthesized supervisor is maximally permissive, as well as controllable, and nonblocking,
- the concept of forcible events from TDESs is used to provide a more comprehensive result, and
- to provide technical proofs, the notion of clock regions of timed automata is adapted in a specific way.

To the best of our knowledge, there is no work in the literature investigating TA RW-based synthesis without abstraction as we do here [2]. Our synthesis technique is close to supervisory control synthesis for EFA. The main differences between EFA and TA are as follows: 1) an EFA deals with a set of variables belonging to a finite domain. However, a TA deals with clock variables, which belong to the infinite set of real-valued numbers, and 2) a TA includes location invariants that force the TA to leave the location before the invariant is violated. This is not the case in EFA. Dealing with real-valued clock variables and location invariants make the synthesis of TA much more complex than the synthesis of EFA. Details are discussed throughout the chapter.

The rest of the chapter is organized as follows. In Section 5.2, the formal definition of TA and the relevant concepts are given. Section 5.3 presents the basic timed supervisory control (TSC) synthesis problem and the proposed solution. In Section 5.4, the basic TSC synthesis problem is generalized to satisfy a given set of control requirements. To verify the results, the proposed method is applied to a rail road crossing system in Section 5.5. Finally, Section 5.6 concludes this chapter.

5.2 Preliminaries

A TA is an FA extended with a finite set of real-valued clocks. To model the timing behavior of TA, the accepting temporal conditions to switch between different modes (locations) or stay in the current one are represented by clock constraints [Alur and Dill 1994; Bengtsson and Yi 2004]. Some of the following definitions have already appeared in Chapter 4.

Definition 5.1 (Clock Constraints [Bengtsson and Yi 2004])**.** Given a finite set of real-valued clocks C, $x \sim n$ and $x - y \sim n$ are atomic clock constraints for any $x, y \in C$, $\sim \in \{<, =, >\}$, and $n \in \mathbb{N}$. Clock constraints are defined as follows: any atomic clock constraint is a clock constraint, and for any two clock constraints φ_1 and φ_2, also $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$ are clock constraints. ■

[2] Rashidinejad et al. [2020b] presents an earlier version of the approach proposed in this chapter.

Instead of writing $x - x = 0$ with $x \in C$ as a clock constraint, we write *true*. Similarly, *false* is written instead of $x - x > 0$.

Definition 5.2 (Clock Valuation)**.** Given a set of clocks C, a clock valuation $u : C \to \mathbb{R}_{\geq 0}$ assigns a real value to each clock $x \in C$. ■

Note that, initially, the valuation of each clock is $\mathbf{0}$, where $\mathbf{0}$ denotes the clock valuation where all the clock variables have value 0.

A clock valuation u satisfies a clock constraint φ, denoted $u \models \varphi$, whenever φ is *true* for the values assigned by u to each clock.

Definition 5.3 (Timed Automaton [Alur and Dill 1994])**.** A timed automaton is a 7-tuple $(C, L, \Sigma, E, L_m, L_0, I)$ where

- C is a finite set of clocks with a non-negative real-value (from $\mathbb{R}_{\geq 0}$). The initial value of each clock variable is always assumed to be 0,
- L is a finite set of locations,
- Σ is a finite set of events,
- E is a finite set of edges with elements e of the form (l_s, σ, g, r, l_t) for which $l_s, l_t \in L$ are the source and target locations, respectively, $\sigma \in \Sigma$, g is the guard, which is a clock constraint, and $r \subseteq C$ is the set of clocks to be reset to 0,
- $L_m \subseteq L$ is the set of marked locations,
- $L_0 \subseteq L$ is the set of initial locations,
- I is a function associating an invariant to each location $l \in L$. An invariant is a clock constraint that needs to be satisfied when the system is in the location. ■

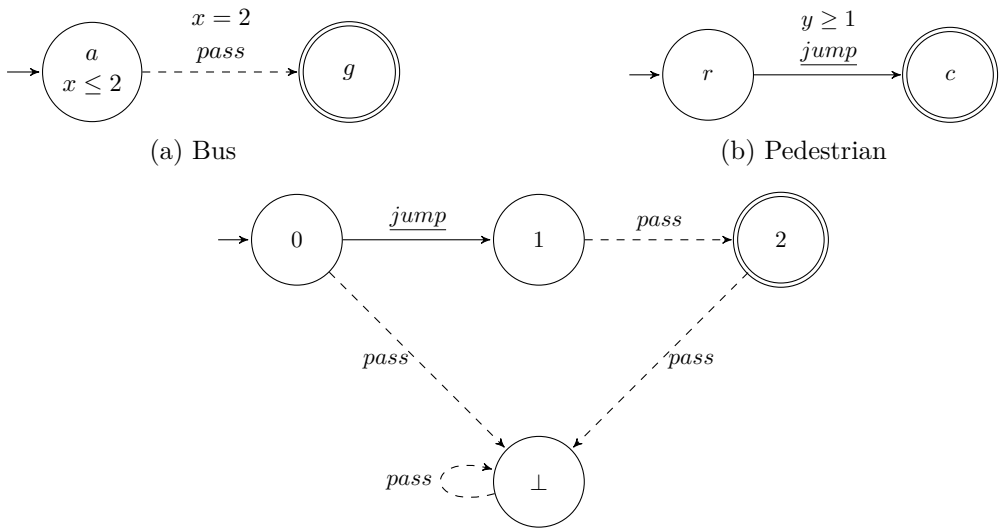
In Bengtsson and Yi [2004], guards are generally given as clock constraints, but invariants are restricted to clock constraints that are downwards closed; $x < n$ or $x \leq n$. In this work, similar to [Alur 1999], both guards and invariants are allowed to be arbitrary clock constraints.

To clarify the problem and illustrate each step of the approach, the bus-pedestrian example from [Brandin and Wonham 1994] is used throughout the chapter.

Example 5.1 (Bus-Pedestrian)**.** Imagine that a bus is headed directly for a pedestrian and will run over him at time $x = 2$ if he does not move. The pedestrian needs an amount of time $y = 1$ to realize his fate, after which he has the chance to jump out of the bus's path. If the pedestrian jumps before the bus passes, he is safe. Figure 5.1 gives the automata, representing the bus, the pedestrian, and the safe behavior of the system. The safe behavior is modeled in such a way that if the pedestrian jumps before the bus passes, then the system goes to a marked state. Otherwise, the system goes to a blocking state.

For TA, we frequently use the following notations:

- the notation . is used to refer to an element of a tuple. For instance, $e.\sigma$ refers to σ from the edge $e \in E$.
- the notation $u[r]$, for a clock valuation u and a reset r, assigns 0 to each clock $x \in r$. The values of $C \setminus r$ stay the same.



(a) Bus (b) Pedestrian

(c) Safe behavior

Figure 5.1: Plant automata from Example 5.1.

- the notation $pred^{\uparrow\delta}$, for a predicate *pred* and the increase $\delta \in \mathbb{R}_{\geq 0}$, replaces all occurrences of the variables $x \in C$ by $x + \delta$. For instance, $(x \geq 3)^{\uparrow\delta}$ gives $x + \delta \geq 3$.
- the meaning of the notation $pred[r]$, for a predicate *pred* and a reset r, is a predicate in which all occurrences of clock variables from r are replaced by zero.
- the notation $Preds(C)$ is used to indicate the set of all predicates over the clock variables.
- the notation P stands for the natural projection operator as defined in Cassandras and Lafortune [2009]; given a language $L \subseteq \Sigma^*$ and an event set $\Sigma' \subseteq \Sigma$: $P_{\Sigma'}(L) := \{w' \in \Sigma'^* \mid \exists w \in L, P_{\Sigma'}(w) = w'\}$.

Here, we only deal with *deterministic* TA [Alur and Dill 1994].

Definition 5.4 (Deterministic TA)**.** A timed automaton $(C, L, \Sigma, E, L_m, L_0, I)$ is deterministic if it has only one initial location $L_0 = \{l_0\}$, and for any pair of edges $e_1, e_2 \in E$, with the same source location ($e_1.l_s = e_2.l_s$) and labeled by the same event ($e_1.\sigma = e_2.\sigma$), the clock constraints are mutually exclusive ($e_1.g \wedge e_2.g = \mathit{false}$). ■

From now on, we only use TA with a single initial location l_0 and consequently represent them by $(C, L, \Sigma, E, L_m, l_0, I)$.

In the examples, TA are depicted graphically. The locations are represented by circles and the edges by arrows from the source location to the target location, labelled with the event, the guard and the reset. The reset of a clock $x \in r$ is denoted by $x := 0$. Invariants of locations are indicated inside the locations. Absence of an invariant in a location represents the invariant that always holds. The initial location is depicted by a dangling incoming arrow, and the marked locations by double circles.

Definition 5.5 (Sub-automaton of a TA)**.** Given a TA $A = (C, L, \Sigma, E, L_m, l_0, I)$, a TA $B = (C, L', \Sigma, E', L'_m, l'_0, I')$ is a sub-automaton of A, denoted $B \subseteq A$, if

- $L' \subseteq L$,
- for all $(l_s, \sigma, g', r, l_t) \in E' : (l_s, \sigma, g, r, l_t) \in E$ for some g such that $g' \Rightarrow g$,
- $L'_m = L_m \cap L'$,
- $l'_0 = l_0$, and
- for all $l \in L'$: $I'(l) \Rightarrow I(l)$. ■

Applications are typically modeled by a network of automata, where each automaton represents a single component or subsystem; compare Figure 5.1. A single automaton representing the network of automata can then be generated as the synchronous product of the constituent automata.

In Alur and Dill [1994] and Bengtsson and Yi [2004], synchronous product of TA is defined under the assumption that the two TA do not share any clock variable. This assumption is relaxed here, and the synchronous product is generalized for TA that may share clocks. To do so, we are inspired from the synchronous product of EFA as defined in Skoldstam et al. [2007].

Definition 5.6 (Synchronous Product of TA)**.** The synchronous product of two TA $G_1 = (C_1, L_1, \Sigma_1, E_1, L_{1m}, l_{10}, I_1)$ and $G_2 = (C_2, L_2, \Sigma_2, E_2, L_{2m}, l_{20}, I_2)$, is given by $G_1||G_2 = (C_1 \cup C_2, L_1 \times L_2, \Sigma_1 \cup \Sigma_2, E_p, L_{1m} \times L_{2m}, (l_{10}, l_{20}), I_p)$, where for each $l_1 \in L_1$ and $l_2 \in L_2$, $I_p(l_1, l_2) = I_1(l_1) \wedge I_2(l_2)$ and E_p is the smallest set that satisfies the following:

- whenever $\sigma \in \Sigma_1 \setminus \Sigma_2$, then for every $(l_{s1}, \sigma, g_1, r_1, l_{t1}) \in E_1$ and $l_2 \in L_2$, $((l_{s1}, l_2), \sigma, g_1, r_1, (l_{t1}, l_2)) \in E_p$
- whenever $\sigma \in \Sigma_2 \setminus \Sigma_1$, then for every $(l_{s2}, \sigma, g_2, r_2, l_{t2}) \in E_2$ and $l_1 \in L_1$, $((l_1, l_{s2}), \sigma, g_2, r_2, (l_1, l_{t2})) \in E_p$.
- whenever $\sigma \in \Sigma_1 \cap \Sigma_2$, then for every $(l_{s1}, \sigma, g_1, r_1, l_{t1}) \in E_1$ and $(l_{s2}, \sigma, g_2, r_2, l_{t2}) \in E_2$, $((l_{s1}, l_{s2}), \sigma, g_1 \wedge g_2, r_1 \cup r_2, (l_{t1}, l_{t2})) \in E_p$. ■

For the bus-pedestrian example, the synchronous product of the bus, pedestrian and the safe behavior automata is shown in Figure 5.2.

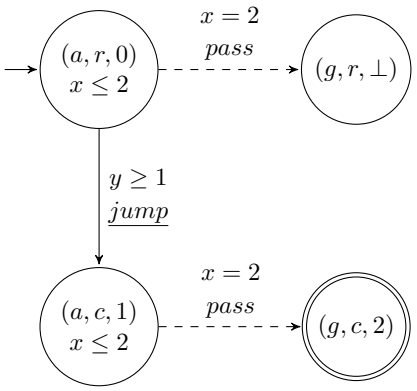


Figure 5.2: Synchronous product of the TA from Example 5.1.

Every TA has an underlying semantic graph [Alur and Dill 1994; Tripakis and Yovine 2001].

Definition 5.7 (Semantic Graph)**.** The semantic graph of a TA $G = (C, L, \Sigma, E, L_m, l_0, I)$, is a labeled graph with a set of states $X \subseteq L \times (C \to \mathbb{R}_{\geq 0})$, consisting of a location l and a clock valuation u such that $(l, u) \in X$ iff $u \models I(l)$. The initial state is $(l_0, \mathbf{0})$ if $\mathbf{0} \models I(l_0)$. Otherwise, the semantic graph is undefined. The semantic graph has the following transitions:

- event transition: from state (l_s, u_s) to state $(l_t, u_s[r])$ labeled by event σ if there is an edge $e = (l_s, \sigma, g, r, l_t)$ such that $u_s \models g$, and $u_s[r] \models I(l_t)$.
- time transition: from state (l, u) to state $(l, u + \Delta)$ labeled with delay $\Delta \in \mathbb{R}_{\geq 0}$ if $u + \delta \models I(l)$ for any δ such that $0 \leq \delta \leq \Delta$. Note that for a valuation u and a real value δ, $u + \delta$ denotes the clock valuation with $(u + \delta)(x) = u(x) + \delta$ for each clock $x \in C$.

Moreover, states (l, u) in the semantic graph with $l \in L_m$ (regardless of the clock valuation u) are marked. A word w in the semantic graph of G is a finite sequence of labels; $w \in (\Sigma \cup \mathbb{R}_{\geq 0})^*$ with ε denoting the empty sequence. A state in the semantic graph of G is called reachable if it can be reached from the initial state via a word. The language of G, denoted $L(G)$, is the set of all words in its semantic graph starting from the initial state. Note that for any $G' \subseteq G$: $L(G') \subseteq L(G)$. ■

Note that since a TA is allowed to have arbitrary clock constraints as invariants, it may be the case that $\mathbf{0} \not\models I(l_0)$. This may happen regarding modeling issues, or through synthesis, where in the latter case, synthesis actually does not result in a supervisor.

Based on the semantic graph, some relevant notions for timed automata are defined.

Definition 5.8 (Nonblockingness)**.** A state in a semantic graph is nonblocking if there exists a path leading from that state to a marked state, i.e., a state (l_t, u_t) with $l_t \in L_m$. A TA is nonblocking if all of the reachable states in its semantic graph are nonblocking. ■

In the rest of the chapter, the plant is given as a TA G represented by $(C, L, \Sigma_G, E_G, L_m, l_0, I_G)$. It is assumed that all events are observable. However, not all of the events might be controllable. The set of events Σ_G is assumed to be partitioned into a set of uncontrollable events Σ_{uc} and a set of controllable events $\Sigma_c = \Sigma_G \setminus \Sigma_{uc}$. Uncontrollable events are events that occur spontaneously in the plant such as disturbances or sensor readings. Controllable events are signals sent to the actuators. In figures of TA, edges labelled by uncontrollable events are indicated by dashed lines, and edges labelled by controllable events are indicated by solid lines. Time passage is uncontrollable by nature. However, it may be preempted by execution of a forcible event $\sigma_f \in \Sigma_{for}$, where $\Sigma_{for} \subseteq \Sigma_G$ (forcible events are underlined in figures). Consequently, considering the semantic graph of a TA, a time transition enabled at a state is considered uncontrollable by default, unless there is also a forcible event transition enabled at that state. Then, the time transition is said to be *preemptable*. Note that a forcible event can be controllable or uncontrollable as discussed in Wonham [2015]. For the bus-pedestrian example, the event *pass* is uncontrollable, and the event *jump* is controllable and forcible.

The following definition of *controllability* for TA with forcible events, is inspired from [Brandin and Wonham 1994].

Definition 5.9 (Controllability of TA with Forcible Events)**.** Given a plant G with uncontrollable events Σ_{uc}, and forcible events Σ_{for}, a TA S is *controllable* w.r.t. G if for all $w \in L(S||G)$ and $\sigma \in \Sigma_{uc} \cup \mathbb{R}_{\geq 0}$, whenever $w\sigma \in L(G)$:

1. $w\sigma \in L(S||G)$, or
2. $\sigma \in \mathbb{R}_{\geq 0}$ and $w\sigma' \in L(S||G)$ for some $\sigma' \in \Sigma_{for}$.
   Property (1) above is the standard controllability property; S cannot disable uncontrollable events that G may generate. However, if a forcible event is enabled, this may preempt the time event, which is captured by Property (2). ■

A supervisor S is called *proper* for a plant G whenever S is controllable w.r.t. G, and the supervised plant $S||G$ is nonblocking.

Definition 5.10 (Maximal Permissivenesss)**.** A proper supervisor S is *maximally permissive* for a plant G, whenever S preserves the largest admissible behavior of G compared to any other proper supervisor S'; for any proper S': $L(S'||G) \subseteq L(S||G)$. ■

As stated in Alur and Dill [1994], the clock valuations of a TA G can be divided into a finite set of clock regions using the definition of region equivalence. Here, we introduce extended clock regions of a TA G, denoted R_G.

Definition 5.11 (Extended Clock Regions of TA)**.** Consider a TA G with a set of clocks C where the the clock ceiling function, $k : C \to \mathbb{N}$ gives the largest natural number that a clock $x \in C$ is bounded to by guards or invariants. Each clock region $r_G \in R_G$ is specified by:

1. for each clock $x \in C$, a single clock constraint of one of the following forms:
   - $x = n$ for some $n \in \{0, \ldots, k(x)\}$,
   - $n - 1 < x < n$ for some $n \in \{1, 2, \ldots, k(x)\}$, or
   - $x > k(x)$
2. for any two different clocks $x, y \in C$, a single clock constraint of one of the following forms:
   - $y - x + k(x) = q$ for some $q \in \{0, \ldots, k(x) + k(y)\}$,
   - $q - 1 < y - x + k(x) < q$ for some $q \in \{1, \ldots, k(x) + k(y)\}$,
   - $y - x + k(x) < 0$, or
   - $y - x + k(x) > k(x) + k(y)$ ■

Note that $k(x)$ does not restrict the value of the clock variable x; it only gives the largest number that x is bounded to by guards or invariants. Considering Figure 5.2, $k(x) = 2$. However, in location $(g, r, \perp)$, the value of x can grow to any real number larger than or equal to 2.

Example 5.2. Figure 5.3 depicts the extended clock regions for a TA with two clock variables x, y, where $k(x) = 2$ and $k(y) = 1$. The clock regions given for the same example in Alur and Dill [1994] are indicated in black.

We call a clock region *unbounded* (dashed areas/lines in Figure 5.3) if it is related to

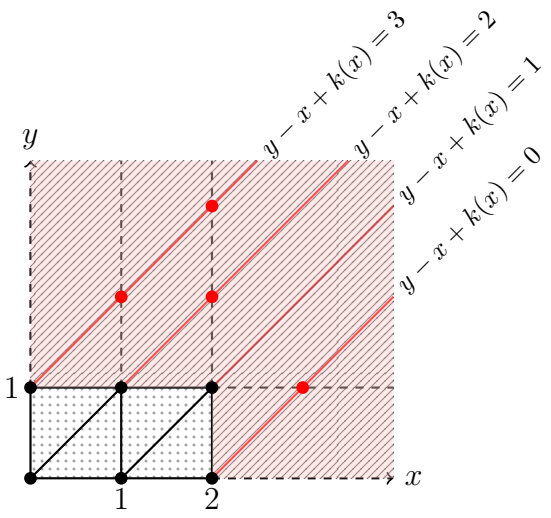


Figure 5.3: Extended clock regions from Example 5.2.

$x > k(x)$ for some $x \in C$. Otherwise, the region is called *bounded* (dotted areas/solid lines in Figure 5.3). Note that although the number of the extended clock regions is more than the number of clock regions, it is still finite because the set of clock regions is finite (see [Alur and Dill 1994] for details), and the extended clock regions include all the bounded regions from the set of clock regions, and it partitions each unbounded region into a finite number of new regions. For instance, in Example 5.2, $0 < x < 1, y > 1$ is an unbounded region that is partitioned into new regions as $0 < x < 1, y > 1, y - x + k(x) < 3$; $0 < x < 1, y > 1, y - x + k(x) = 3$; and $0 < x < 1, y > 1, y - x + k(x) > 3$.

Definition 5.12 (G-Clock Constraint)**.** Consider a plant G with a set of clocks C, the clock ceiling function $k : C \to \mathbb{N}$, and the set of regions R_G. A clock constraint φ is called a *G-clock constraint* whenever all the atomic constraints of φ are bounded by $k(x)$ for all $x \in C$. ■

Clearly, for any two G-clock constraints φ_1 and φ_2, $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$ are G-clock constraints.

Based on the extended clock regions, we are now able to discriminate the regions that satisfy a G-clock constraint. Let us consider Example 5.2 again. Given a G-clock constraint $\varphi = x - y > 2$, there does not exist a set of clock regions satisfying φ based on the definition of clock regions in Alur and Dill [1994]. However, considering Definition 5.11, $y = 0, x > 2, y - x + k(x) < 0$; $0 < y < 1, x > 2, y - x + k(x) < 0$; $y = 1, x > 2, y - x + k(x) < 0$; and $y > 1, x > 2, y - x + k(x) < 0$ are the extended clock regions satisfying φ. This discrimination will be the basis to prove the termination and correctness of the proposed algorithms.

Moreover, it is assumed that there exists a function Z mapping a G-clock constraint φ to the maximal set of regions from R_G such that for any region $r_G \in Z(\varphi)$, and for any valuation u represented by r_G, denoted $u \in r_G$, $u \models \varphi$. For any two G-clock constraints φ_1 and φ_2, Z necessarily satisfies the following properties:

- $Z(\varphi_1 \wedge \varphi_2) = Z(\varphi_1) \cap Z(\varphi_2)$ and $Z(\varphi_1 \vee \varphi_2) = Z(\varphi_1) \cup Z(\varphi_2)$.
- Whenever $Z(\varphi_1) = Z(\varphi_2)$, φ_1 and φ_2 represent the same G-clock constraint.

Also, for the clock constraints represented by *true* and *false*, the mapping gives R_G, and $\varnothing$, respectively.

5.3 Basic TSC Synthesis

The *Basic TSC Synthesis Problem* is defined as follows.

Problem Statement: Given a plant model G as a TA, the objective is to synthesize a timed supervisor S, also as a TA, such that

- S is controllable w.r.t. G,
- $S||G$ is nonblocking, and
- S is maximally permissive w.r.t. G.

Considering the bus-pedestrian example, a supervisor is required to avoid reaching the blocking location $(g, r, \perp)$ in Figure 5.2. The objective is to provide a supervisory control synthesis approach that does not need an abstraction. The synthesized supervisor should respect controllability (Definition 5.9), nonblockingness (Definition 5.8), and be maximally permissive (Definition 5.10).

To synthesize such a supervisor, it is needed to determine the states (l, u) in the semantic graph that should be made unreachable, referred to as *bad states*. These are the following types of states: 1) states that are blocking and should be avoided to take care of nonblockingness, and 2) states that lead to a bad state through an uncontrollable event or a time transition that cannot be preempted; these states should be avoided to respect controllability as well as nonblockingness. As the synthesis algorithm should not involve any abstraction, we need to determine the clock valuations for which a location of a TA is a bad state (in the semantic graph). For this purpose, we start by determining the clock valuations for which a location is nonblocking, referred to as the "nonblocking predicate" of a location. Based on the nonblocking predicate, a "bad state predicate" is associated to each location determining the clock valuations for which the location is mapped to a bad state in the semantic graph.

5.3.1 Nonblocking Condition

Given a plant G, Algorithm 5.1 associates a nonblocking predicate $N(l)$ to each location $l \in L$. Initially (line 3), $N^i(l)$ with $i = 0$ is set to $I_G(l)$ if l is a marked location, and to *false* otherwise. The nonblocking predicate of each location is updated (line 6) to $N^{i+1}(l)$ based on:

① the current nonblocking predicate $N^i(l)$,

② the condition for any outgoing edge (l, σ, g, r, l') to lead to a nonblocking location (an event transition leading to a nonblocking state in the semantic graph), and

③ the condition to stay (for some time delay $\delta \leq \Delta$) in a nonblocking location as long as the invariant is satisfied (represented by a time transition leading to a nonblocking state in the semantic graph).

This iterates until a fix-point is reached where the nonblocking predicate stays the same for all locations (line 8).

Algorithm 5.1 Nonblocking Predicate (*NBP*)

Input: $G = (C, L, \Sigma_G, E_G, L_m, l_0, I_G)$
Output: $N : L \to Preds(C)$

1: $i := 0$
2: **for** $l \in L$ **do** $N^0(l) := \begin{cases} I_G(l), & \text{if } l \in L_m, \\ false, & \text{otherwise} \end{cases}$
3: **end for**
4: **repeat**
5: **for** $l \in L$ **do**

$$N^{i+1}(l) := \overbrace{N^i(l)}^{①} \vee \overbrace{\bigvee_{l \xrightarrow{\sigma,g,r} l'} (g \wedge I_G(l')[r] \wedge N^i(l')[r])}^{②} \vee$$

$$\overbrace{\exists \Delta\ N^i(l)^{\uparrow \Delta} \wedge \forall \delta \leq \Delta\ I_G(l)^{\uparrow \delta}}^{③}$$

6: **end for**
7: $i := i + 1$
8: **until** $\forall l \in L\ N^i(l) = N^{i-1}(l)$
9: **for** $l \in L$ **do** $N(l) := N^i(l)$
10: **end for**

Algorithm 5.1 follows the same steps as presented for the nonblocking predicate of EFA in Ouedraogo et al. [2011] with the following adjustments (indicated in red in Algorithm 5.1):

1. The initial nonblocking condition for marked locations is set to the location invariant $I_G(l)$ instead of *true*. This is to take into account the invariants of the marked locations.
2. In the update (line 6), the invariant of the target location is added to the second term to guarantee that the invariant of the target location is satisfied upon entering that location.
3. The third term is added to take into account the time transitions in the semantic graph of the TA that may be used for reaching a nonblocking state.

Property 5.1 (*NBP* Termination)**.** Given a plant G with a set of locations L and a set of regions R_G; Algorithm 5.1 terminates.

Proof. See Appendix D.2.1. ■

Property 5.2 (*NBP* and Nonblocking States)**.** Given a plant G and $NBP(G)$: for any (l, u) in (the semantic graph of) G, (l, u) is a nonblocking state iff $u \models N(l)$, where $N = NBP(G)$.

Proof. See Appendix D.2.2. ■

Example 5.3 (Nonblocking Predicate for Bus-Pedestrian)**.** Let us consider the bus-pedestrian from Example 5.1. The result of Algorithm 5.1 is given in Table 5.1. The conditions for locations $(g, r, \bot)$ and $(g, c, 2)$ are left out, as they are *false* and *true* respectively, for all iterations. The condition $x \leq 2 \wedge (y \geq 1 \vee x - y \leq 1)$ is equivalent to $x \leq 2 \wedge x - y \leq 1$.

Table 5.1: Nonblocking predicate for bus-pedestrian.

	N	
i	**Loc** $(a, r, 0)$	**Loc** $(a, c, 1)$
0	*false*	*false*
1	*false*	$x = 2$
2	$x = 2 \wedge y \geq 1$	$x \leq 2$
3	$x \leq 2 \wedge (y \geq 1 \vee x - y \leq 1)$	$x \leq 2$
4	$x \leq 2 \wedge x - y \leq 1$	$x \leq 2$

5.3.2 Bad State Condition

Given a plant G, and the nonblocking predicate computed by Algorithm 5.1, Algorithm 5.2 associates a bad state predicate $B(l)$ to each location $l \in L$.

Initially, $B^i(l)$ with $i = 0$ is set to the logical negation of $N(l)$ for each location $l \in L$ (line 3) because these characterize the blocking states. Then, the bad state predicate of each location is updated to $B^{j+1}(l)$ (line 6) based on

④ the previous bad state predicate $B^j(l)$,

⑤ the condition of any outgoing edge (l, σ, g, r, l') labeled by an uncontrollable event $\sigma \in \Sigma_{uc}$ to lead to a bad state (an uncontrollable event transition leading to a bad state in the semantic graph), and

⑥ the condition of staying in a bad state for some time delay $\delta \leq \Delta$ as long as the invariant is satisfied for all the clock variables and while there is no forcible event able to preempt time for any $\delta' \leq \delta$ (an uncontrollable time transition leading to a bad state in the semantic graph).

This iterates until a fix-point is reached where the bad state predicate stays the same for all locations (line 8).

Algorithm 5.2 Bad State Predicate (BSP)

Input: $G = (C, L, \Sigma_G, E_G, L_m, l_0, I_G), NBP(G)$
Output: $B : L \to Preds(C)$

1: $j := 0$
2: **for** $l \in L$ **do** $B^0(l) := \neg N(l)$
3: **end for**
4: **repeat**
5: **for** $l \in L$ **do**

$$B^{j+1}(l) := \overbrace{B^j(l)}^{④} \vee \overbrace{\bigvee_{l \xrightarrow[\sigma \in \Sigma_{uc}]{\sigma,g,r} l'} \Big(g \wedge I_G(l')[r] \wedge B^j(l')[r]\Big)}^{⑤} \vee$$

$$\overbrace{\exists \Delta\, B^j(l)^{\uparrow \Delta} \wedge \forall \delta \leq \Delta \Big(I_G(l)^{\uparrow \delta} \wedge \forall \delta' \leq \delta \quad \neg \bigvee_{l \xrightarrow[\sigma_f \in \Sigma_{for}]{\sigma_f,g,r} l'} (g^{\uparrow \delta'} \wedge I_G(l')^{\uparrow \delta'}[r] \wedge \neg B^j(l')^{\uparrow \delta'}[r])\Big)}^{⑥}$$

6: **end for**
7: $j := j + 1$
8: **until** $\forall l \in L\ B^j(l) = B^{j-1}(l)$
9: **for** $l \in L$ **do** $B(l) := B^j(l)$
10: **end for**

The differences (indicated in red) between Algorithm 5.2 and the bad state condition of EFA presented by [Ouedraogo et al. 2011] are as follows; 1. The invariant of the target location is considered to determine if the uncontrollable transition should exist in the semantic graph. 2. The third term takes into account the non-preemptable time transitions leading to a bad state.

Property 5.3 (BSP Termination)**.** Given a plant G with the set of locations L, set of regions R_G, and $NBP(G)$; Algorithm 5.2 terminates.

Proof. See Appendix D.2.3. ■

Property 5.4 (BSP and Bad States)**.** Given a plant G and $NBP(G)$: for any (l, u) in (the semantic graph of) G, (l, u) is a bad state iff $u \models B(l)$, where $B = BSP(G, NBP(G))$.

Proof. See Appendix D.2.4. ■

Example 5.4 (Bad State Predicate for Bus-Pedestrian)**.** By applying Algorithm 5.2 on the bus-pedestrian example, the bad state predicate of locations $(a, r, 0)$ and $(a, c, 1)$ are

obtained as in Table 5.2. The bad state predicates for $(g, r, \bot)$ and $(g, c, 2)$ are *true* and *false*, respectively.

Table 5.2: Bad state predicate for bus-pedestrian.

	B	
j	**Loc** $(a, r, 0)$	**Loc** $(a, c, 1)$
0	$x > 2 \vee x - y > 1$	$x > 2$
1	$x \geq 2 \vee x - y > 1$	$x > 2$
2	$x \geq 2 \vee x - y > 1$	$x > 2$

5.3.3 Synthesis

Figure 5.4 gives an overview of the synthesis procedure. As indicated in the figure, there are two loops: 1. guard adaptation (Loop-1) considers how the supervisor can affect the controllable events, and 2. invariant adaptation (Loop-2) considers how the invariants can be modified using the concept of forcible events.

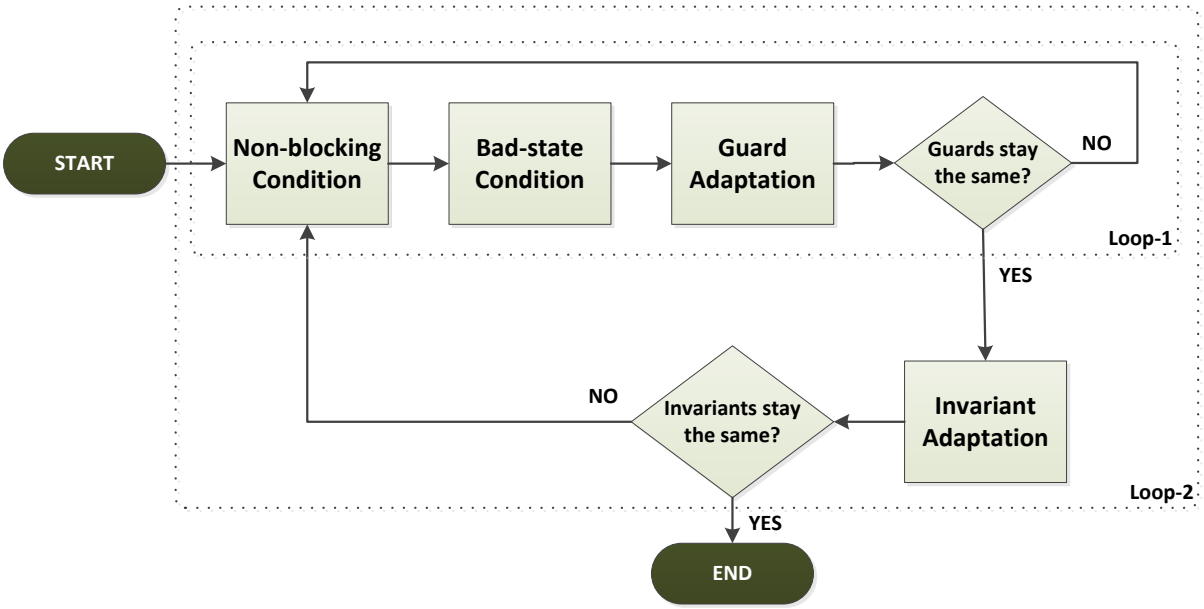


Figure 5.4: An overview of the synthesis procedure.

Guard adaptation

Consider Figure 5.4, in Loop-1 the guards are adapted to obtain a supervisor that prevents the bad states. For this purpose, the guard of each edge (l, σ, g, r, l') labeled by a controllable event $\sigma \in \Sigma_c$ is adjusted to become $(l, \sigma, g \wedge \neg B(l')[r], r, l')$.

Invariant adaptation

So far, forcible events have not been taken into account. The effect of forcible events preempting time events is taken into account in the invariant adaptation (Loop-2). The invariant of a location $l \in L$ can be changed only if there exists an edge labeled by a forcible event $\sigma_f \in \Sigma_{for}$ starting from l. In this case, the invariant is adapted to prevent reaching the bad states as follows:

$$I(l) := I(l) \wedge \neg B(l).$$

Synthesis Algorithm

Algorithm 5.3 is the synthesis algorithm. For a TA G with a set of uncontrollable events Σ_{uc}, and a set of forcible events Σ_{for}, it results in $S = (C, L, \Sigma_G, E_S, L_m, l_0, I_S)$. The notation $F_S(l) = \{e \in E_S \mid e.l_s = l,\ e.\sigma \in \Sigma_{for},\ e.g \text{ is satisfiable}\}$ gives the set of edges of S starting from location l and labeled by a forcible event. The algorithm starts with $S = G$. As indicated in Figure 5.4, in the inner loop (lines 9-16), the guards of edges labeled by controllable events are adapted until a fix-point is reached. In the outer loop (lines 7-25), the invariants of locations where there exists an edge labeled by a forcible event are adapted until a fix-point is reached. Otherwise, the synthesis goes back to Loop-1 (guard adaptation). Note that if the invariant of a location l is adapted, and in some later iteration the guard of an edge labeled by the forcible event becomes false, then the invariant should be set back to its original $I_G(l)$. This is captured in line 21.

Given a plant G, in the case that $u_0 \models B(l_0)$, with B as the result of Algorithm 5.2 for $TSCS(G)$ and $NBP(TSCS(G))$, then $TSCS(G)$ is undefined. In the rest of the chapter, it is assumed that $u_0 \not\models B(l_0)$ for any given plant G.

Property 5.5 ($TSCS$ Termination)**.** Given a plant G; Algorithm 5.3 terminates.

Proof. See Appendix D.2.5. ■

Property 5.6 ($TSCS(G)$ is a TA)**.** Given a plant G, $S = TSCS(G)$ is a TA.

Proof. See Appendix D.2.6. ■

Property 5.7 ($TSCS(G)$ is a subautomaton of G)**.** Given a plant G: $TSCS(G) \subseteq G$.

Proof. See Appendix D.2.7. ■

According to Property 5.7, $TSCS(G)||G = TSCS(G)$.

Property 5.8 (Algorithm Correctness)**.** Given a plant G and the supervisor $S = TSCS(G)$: for any reachable state (l, u) (in the semantic graph) of S: $u \not\models B(l)$, where $B = BSP(S, NBP(S))$.

Proof. See Appendix D.2.8. ■

The following theorems summarize the main results.

Algorithm 5.3 Timed supervisory control synthesis (*TSCS*)

Input: $G = (C, L, \Sigma_G, E_G, L_m, l_0, I_G)$, Σ_{uc}, Σ_c, Σ_{for}
Output: $S = (C, L, \Sigma_G, E_S, L_m, l_0, I_S)$

$S := G$
$n := 0$
for $e \in E_S$, $e = (l, \sigma, g, r, l')$ **do** $e.g^0 := e.g$
end for
for $l \in L$ **do** $I_S^0(l) := I_G(l)$
end for
repeat ▷ Loop-2: Invariant Adaptation
  $m := 0$
  repeat ▷ Loop-1: Guard Adaptation
    $N^{n,m} := NBP(S)$
    $B^{n,m} := BSP(S, N^{n,m})$
    for $e \in E_S$ such that $e.\sigma \in \Sigma_c$ **do**
      $e.g^{m+1} := e.g^m \wedge \neg B^{n,m}(l')[r]$
    end for
    $m := m + 1$
  until $\forall e \in E_S\ e.g^m = e.g^{m-1}$
  for $e \in E_S$ **do** $e.g := e.g^m$
  end for
  for $l \in L$ **do**
    if $F_S(l) \neq \varnothing$ **then** $I_S^{n+1}(l) := I_S^n(l) \wedge \neg B^{n,m}(l)$
    else $I_S^{n+1}(l) := I_G(l)$
    end if
  end for
  $n := n + 1$
until $\forall l \in L\ I_S^n(l) = I_S^{n-1}(l)$
for $l \in L$ **do** $I_S(l) := I_S^n(l)$
end for

Theorem 5.1 (Controllability)**.** Given a plant G with uncontrollable events Σ_{uc} and forcible events Σ_{for}, and the supervisor $S = TSCS(G)$: S is controllable w.r.t. G.

Proof. See Appendix D.2.9. ■

Theorem 5.2 (Nonblockingness)**.** Given a plant G and the supervisor $S = TSCS(G)$: the supervised plant $S||G$ is nonblocking.

Proof. See Appendix D.2.10. ■

Theorem 5.3 (Maximal Permissiveness)**.** Given a plant G and the supervisor $S = TSCS(G)$: S is maximally permissive for G.

Proof. See Appendix D.2.11. ■

Example 5.5 (Supervisor Synthesis for Bus-Pedestrian)**.** Let us apply Algorithm 5.3 to the bus-pedestrian from Example 5.1. Initially, S is set to the plant depicted in Figure 5.2. First, the guard of the edge labeled by the controllable event *jump* is modified to $y \geq 1 \wedge x \leq 2$. Since $N^{1,0} = N^{0,0}$ and also $B^{1,0} = B^{0,0}$, $e.g^1 = e.g^0$, and the inner loop stops. Next, for $l_0 = (a, r, 0)$, the invariant is adapted to $x \leq 2 \wedge x < 2 = x < 2$. Since $N^{1,1} = N^{1,0}$ and also $B^{1,1} = B^{1,0}$, $I_S^1(l_0) = I_S^0(l_0)$ and the outer loop also terminates. The synthesized supervisor is depicted in Figure 5.5.

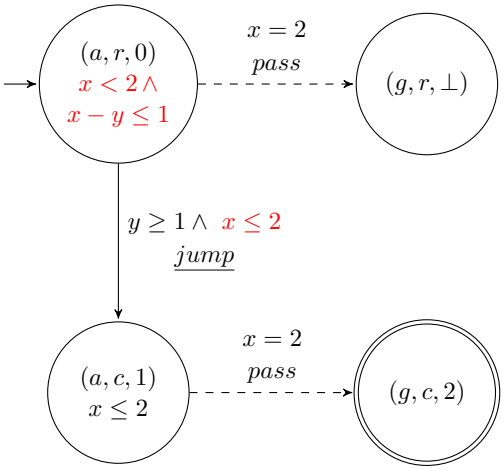


Figure 5.5: Supervisor for bus-pedestrian from Example 5.1.

Remark. Invariant adaptation can highly affect the synthesis result. Consider Example 5.5, Algorithm 5.3 does not result in a supervisor without invariant adaptation. However, if the TA has no forcible event, time transitions are always uncontrollable and the synthesis procedure can be adjusted as follows:

1. the update of the bad state predicate (Algorithm 5.2-line 6) simplifies to

$$B^{j+1}(l) := B^j(l) \vee \bigvee_{\substack{l \xrightarrow{\sigma,g,r} l' \\ \sigma \in \Sigma_{uc}}} \Big(g \wedge I_G(l')[r] \wedge B^j(l')[r]\Big) \vee$$
$$\exists \Delta\ B^j(l)^{\uparrow \Delta} \wedge \forall \delta \leq \Delta\ I_G(l)^{\uparrow \delta}$$

   where the last part of ⑥ is removed, and

2. the algorithm ends after the inner loop indicated in Figure 5.4 since guard adaptation is the only modification that can be applied through synthesis.

5.4 Requirement Automata

To generalize the method to a wider class of applications, we solve the TSC synthesis problem for a given set of control requirements. It is assumed that an allowed behavior of G is denoted by the timed automaton $R = (C_R, Q, \Sigma_R, E_R, Q_m, q_0, I_R)$, where $\Sigma_R \subseteq \Sigma_G$ and $C_R \cap C = \emptyset$. Since most control requirements are defined to provide safety of a plant, we call a supervised plant $SP = S||G$ *safe* if it satisfies the control requirement R.

Definition 5.13 (Safety)**.** Given a plant G and a control requirement R, a TA S with event set Σ_S is safe w.r.t. G and R if $P_{\Sigma_{SP}\cap\Sigma_R}(L(S||G)) \subseteq P_{\Sigma_{SP}\cap\Sigma_R}(L(R))$ with $\Sigma_{SP} = \Sigma_S \cup \Sigma_G$. ■

Requirement automata can be considered in synthesis by being transferred into the plant using synchronous product. However, if a requirement automaton is not controllable (Definition 5.9), then it is necessary to let the supervisor know about the uncontrollable events that are disabled by a given requirement. To take care of this issue, a requirement automaton R is made complete. *Completion* was first introduced in Flordal et al. [2007] for DESs, where the requirement automaton R is made complete as $R^\perp$ in terms of uncontrollable events. By applying the synthesis on $G||R^\perp$, all original controllability problems in $G||R$ are translated to blocking issues. To solve the blocking issues, synthesis still takes the controllability definition into account. Inspired from [Flordal et al. 2007], we present the completion of a TA.

Definition 5.14 (TA Completion)**.** Given a TA $R = (C_R, Q, \Sigma, E_R, Q_m, q_0, I_R)$, the complete automaton $R^\perp$ is defined as $R^\perp = (C_R, Q \cup \{q_d\}, \Sigma, E_R^\perp, Q_m, q_0, I_R)$, where $q_d \notin Q$, $I_R(q_d) = \mathit{true}$ and $I_R(q) = I(q)$ for all $q \in Q$, and for every $q_s \in Q$, $\sigma \in \Sigma_{uc}$:

$$E_R^\perp = E_R \cup \{(q_s, \sigma, g^\perp, \{\}, q_d) \mid (q_s, \sigma, g, r, q_t) \in E_R\},$$

where $g^\perp = \neg\Big(\bigvee_{e\in E_R, e.q_s=q_s, e.\sigma=\sigma} e.g \wedge I_R(e.q_t)[e.r]\Big)$. ■

To synthesize a supervisor, Algorithm 5.3 is applied on $G||R^\perp$. The obtained supervisor is already guaranteed to be controllable, maximally permissive, and it results in a nonblocking supervised plant. Theorem 5.4 shows that the supervised plant is safe as well.

Theorem 5.4 (Safety)**.** Given a plant G, a set of control requirements R, and the supervisor $S = \mathit{TSCS}(G||R^\perp)$: S is safe for G w.r.t. R.

Proof. See Appendix D.2.12. ■

In general, there can be a set of control requirements $\{R_1, R_2, \ldots, R_n\}$ given for a plant. In that case, the allowed behavior of G, is determined by the synchronous product of all requirement automata; $R = R_1||R_2||\ldots||R_n$. Since completion distributes over synchronous product, $R^\perp$ can be computed either as $R_1^\perp||R_2^\perp||\ldots||R_n^\perp$, or $(R_1||R_2||\ldots||R_n)^\perp$.

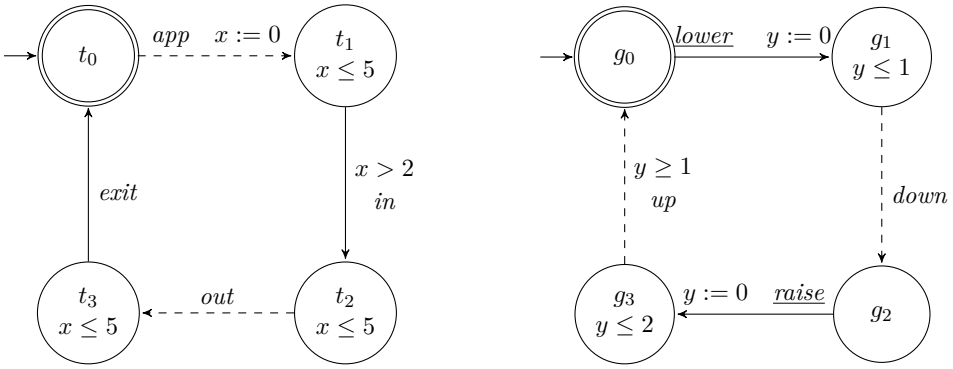
5.5 Case Study

In this section, we consider the verification example from [Alur 1999; Alur and Dill 1994] and modify it for synthesis. The TA representing the train and gate are depicted in Figure 5.6. The system in Alur [1999] and Alur and Dill [1994] also involves an automatic controller, depicted in Figure 5.7, to open and close the gate in a railroad crossing. The control requirements for the train-gate-controller system are as follows [Alur and Dill 1994]:

- Safety requirement: whenever the train is inside the gate, the gate should be closed.
- Liveness requirement: the gate is never closed for more than 10 time units.

In [Alur 1999; Alur and Dill 1994], the system is assessed to be safe by analysing the timing constraints: they say that with the (random) gate-controller, that is part of the system, the event *lower* always precedes the event *in*, so the system is always safe. We do not consider such a controller to already be given as a part of the system. We synthesize a supervisor that is correct-by-construction, and more importantly this supervisor guarantees controllability, nonblockingness, and maximal permissiveness.

The models of train and gate are taken directly from [Alur 1999; Alur and Dill 1994]. The events *app* and *out* for the train, and the events *down* and *up* for the gate are assumed to be uncontrollable. Moreover, the events *raise* and *lower* of the gate are assumed to be forcible.



(a) Train (b) Gate

Figure 5.6: Train-gate system.

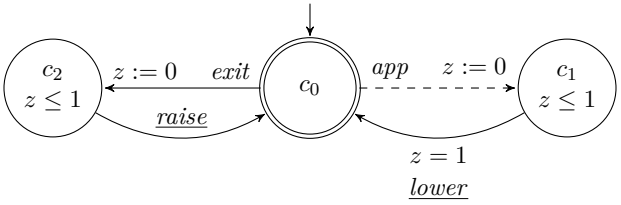
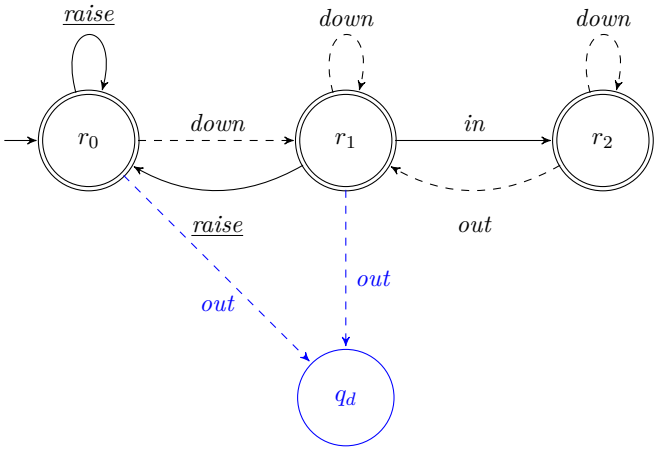


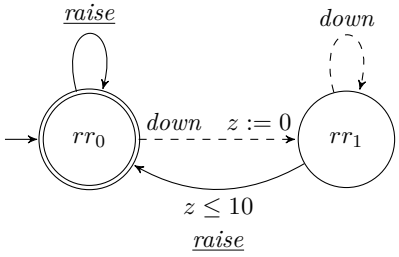
Figure 5.7: Gate-controller from [Alur 1999; Alur and Dill 1994].

The safety requirement is represented by the TA in Figure 5.8a, where the blue location and edges are added to make the TA complete. The liveness requirement is represented by the TA in Figure 5.8b. The liveness requirement does not need completion as the uncontrollable event *down* is enabled at both states of the automaton.

The supervisor synthesized by Algorithm 5.3 for the train-gate and control requirements is given in Figure 5.9. In this figure, the synchronous product of the train-gate and control requirements is indicated in black and the adaptations made by the supervisor in red.



(a) Safety



(b) Liveness

Figure 5.8: Requirements for train-gate system.

5.6 Conclusions

In this chapter, we propose a synthesis algorithm for timed automata (TA) with a set of forcible events. The algorithm is directly applicable on TA without abstracting them to finite state automata. The objective is to avoid blocking states. To take care of controllability, not only the blocking states but also the states from which a blocking state is reachable in an uncontrollable manner (referred to as bad states) should be avoided. The bad states are determined using nonblocking and bad state predicates associated to each location. The modifications made through synthesis are as follows: 1. guard adaptation of edges labeled by controllable events, and 2. invariant adaptation of locations from which there exists an edge labeled by a forcible event. Based on the notion of extended clock regions, it is proven that the synthesized supervisor satisfies nonblockingness, controllability, and maximal permissiveness. To generalize, we solve the problem for a given set of control (safety) requirements modeled as TA. We guarantee that the synthesized supervisor satisfies controllability, nonblockingness, maximal permissiveness, and safety. Finally, the results are verified by applying the method on a case study.

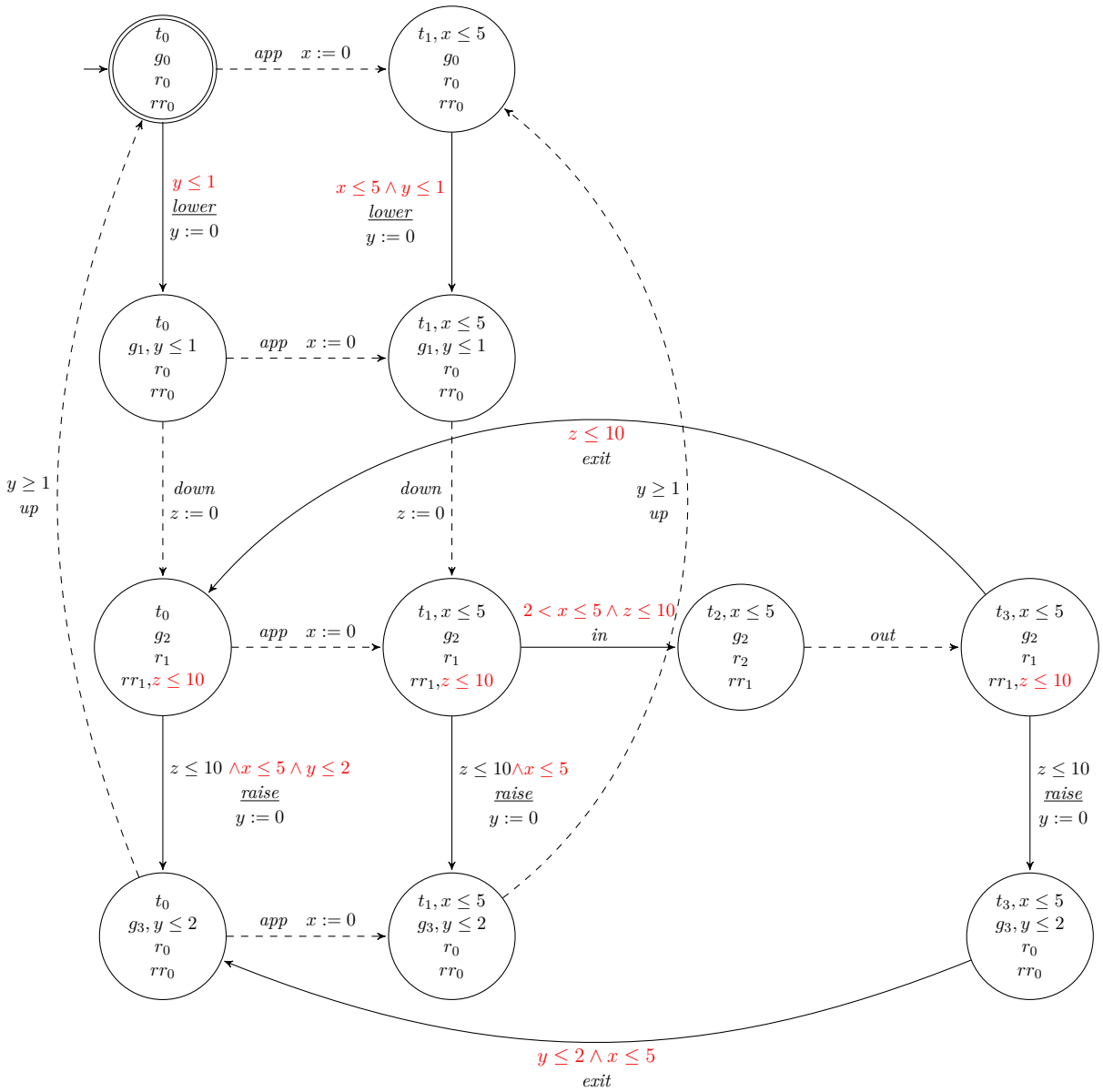


Figure 5.9: Synthesized supervisor for train-gate and control requirements. Edges with guards equal to *false* and locations reached by them have been removed.

Chapter 6

Supervisory Control of Discrete-Event Systems under Attacks

Due to network-based communications, cyber-physical systems face the risks of cyber-attacks, which might result in catastrophic damage. This chapter provides an overview of existing approaches that prevent damage caused by cyber-attacks in the supervisory control of cyber-physical systems. The objective is to identify missing pieces, possible links and determine new directions for further research to extend current practices. For this purpose, first, we classify the current research works under a framework consisting of three dimensions: 1) the communication channel where an attack can happen, 2) the attack impact on the transmitted data, and 3) the mechanism to prevent damage. This classification will then help us to compare the existing techniques and investigate how they can be improved to get closer to the ideal scenario where most kinds of attacks can be handled so that the closed-loop system remains safe.

6.1 Introduction

Cyber-physical systems (CPSs) are the integration of computation, networking and physical processes. CPSs can be found in many areas varying from manufacturing and chemical processes to aerospace and healthcare systems [Shi et al. 2011]. Network-based communications between the cyber part and the physical part in CPSs raises the need to deal with

This chapter is based on Rashidinejad et al. [2019b].

cyber security, especially for applications where an attack might put the system and human safety at risk such as in smart grids or water networks [Amin et al. 2010; Ciancamerla et al. 2014; Eliades and Polycarpou 2010; Nicolaou et al. 2018; Pasqualetti et al. 2011; Sridhar et al. 2011; Teixeira et al. 2010; Zhu and Başar 2011]. For this reason, security of CPSs has gained a lot of attention in recent years [Cardenas et al. 2008; Cardenas et al. 2009; Fawzi et al. 2014; Pasqualetti et al. 2013].

In this chapter, we focus on works investigating cyber-attacks in the supervisory control of a CPS modeled as a discrete-event system (DES). A DES is a dynamical system defined over a discrete set of states with state transitions that are correlated to the occurrences of events [Cassandras and Lafortune 2009]. Supervisory control theory has been developed for DESs to ensure safety and progress properties of the closed-loop control systems [Ramadge and Wonham 1987]. Figure 6.1 depicts a typical supervisory control system[2] under attacks. In an active attack, the attacker's goal is to inflict damage on the system. This is in contrast to a passive attack, where the attacker's goal is to learn secrets about the system [Uma and Padmavathi 2013].

The sensor data gathered from the plant are observed by the supervisor through the observation channel, and control commands are transmitted from the supervisor to the actuators in the plant through the control channel. Notations Σ, $\Sigma_o \subseteq \Sigma$, and $\Sigma_c \subseteq \Sigma$ stand for the set of events executed in the plant, events that are observable by the supervisor and events that are controllable by the supervisor, respectively. The control commands issued by the supervisor are control patterns in Γ, where $\Gamma := \{\gamma \subseteq \Sigma \mid \Sigma_{uc} \subseteq \gamma\}$ and $\Sigma_{uc} = \Sigma \setminus \Sigma_c$ denotes the set of uncontrollable events. Both the observation channel and the control channel may introduce the risk of cyber-attacks. An attacker can corrupt a subset of events (vulnerable events) transmitted from sensors to the supervisor (observable events) and from the supervisor to the actuators in the plant (controllable events). Observable vulnerable events and controllable vulnerable events are denoted by $\Sigma_{ov} \subseteq \Sigma_o$ and $\Sigma_{cv} \subseteq \Sigma_c$, respectively. When attacked, observable vulnerable events or the empty string ϵ can be replaced by a word $w \in \Sigma_{ov}^*$. For a control command $\gamma \in \Gamma$, on the other hand, controllable vulnerable events can only be erased or replaced by other controllable vulnerable events to form a new control command $\gamma' \in \Gamma$ so that $\gamma' \setminus \Sigma_{cv} = \gamma \setminus \Sigma_{cv}$.

The attacker in Figure 6.1 has no extra sensors for observing the execution of the plant, as we assume that the attacker only listens to the communication channels that are vulnerable to attacks. In particular, we assume the attacker has full observation on the data transmitted on the communication channels that are vulnerable to attacks. In addition, we assume the supervisor sends a new control command to the plant whenever it receives a piece of information, i.e., the occurrence of observable events, from the observation channel. There are many other possibilities. For example, the attacker can place additional sensors for observing the execution of the plant, or the attacker has only partial access to the communication channels, or a combination thereof. Also, the supervisor does not need to send a control command each time when it observes an occurrence of observable events. Different assumptions will lead to variations of the architecture given in Figure 6.1.

[2]There are various architectures that are largely incompatible; consequently, it is almost impossible to show all these architectures within a single diagram. Some generalizations and alternatives will also be discussed and explained throughout the chapter.

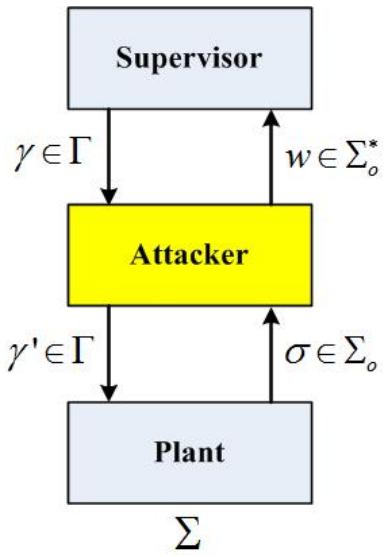


Figure 6.1: Supervisory control system under attacks.

Here, we analyze attack-prevention techniques in supervisory control of CPSs represented by DESs, that are available in the literature, and pinpoint potential directions for future research. For this purpose, first, we provide a framework to classify the current approaches. This classification enables us to compare the available techniques and find areas that need further improvements. Moreover, we discuss some relevant works dealing with similar issues in a supervisory control system, such as faults and various communication problems. Finally, we outline new research lines and conclude the chapter.

6.2 Framework

The attacks studied in this framework are mainly active attacks, where the attacker's goal is to inflict damage rather than learning secrets about the system. Based on the works presented in the literature that discuss the security of a supervisory control system under (active) attacks, we classify them based on the following dimensions:

- the attack location,
- the attack impact on transmitted data,
- the security mechanism.

The first two are related to the characteristics of the attacks, while the last one is related to the different mechanisms by which a supervisor can deal with or prevent the attacks. In the next subsections, these three dimensions are discussed in more detail. As we will soon see, the classification is far from complete and there are many dimensions that we have to omit, due to the limited number of research works in this area. Many aspects that we do not classify are discussed in Section 6.6, including passive attacks.

6.2.1 Attack Location

As Figure 6.1 shows, for a closed-loop supervisory control system, an attack may occur in the observation channel (known as a sensor attack), in the control channel (known as an actuator attack) or in both observation and control channels (as the most realistic case) [Carvalho et al. 2018; Lima et al. 2018]. Due to the different direct impacts that sensor and actuator attacks can make on the system, most of the existing works investigate them separately. Therefore, the location where an attack may occur is considered as a dimension to classify the current works.

6.2.2 Attack Impact on Transmitted Data

In the security of cyber-physical systems, there are three properties that a system should possess for it to work properly, i.e., confidentiality, integrity and availability [Cardenas et al. 2008]. In the literature, different types of cyber-attacks have been classified based on the properties that they threaten [Cardenas et al. 2008; Teixeira et al. 2012]. For instance, deception attacks compromise the integrity, while denial of service attacks compromise the availability of the system [Pasqualetti et al. 2013]. Considering Figure 6.1, no matter what the type of the attack is, a sensor attack changes an event $\sigma \in \Sigma_{ov}$ in one of the following ways: deletion, insertion or replacement, where replacement can be viewed as composition of deletion and insertion. For instance, consider the example from [Lima et al. 2018] where $\Sigma = \Sigma_o = \{a, b, c\}$, $\Sigma_{ov} = \{a, c\}$ and the word $w = abba$ executed in the plant. Then, considering all possibilities of deletion, insertion, and replacement under a sensor attack, the corrupted word belongs to $\{a, c\}^* b \{a, c\}^* b \{a, c\}^*$. An actuator attacker may affect a controllable vulnerable event $\sigma \in \Sigma_{cv}$ in one of the following ways: enablement or disablement. An actuator enablement attack can overwrite the supervisor's disablement action (for a vulnerable event) with an enablement action, and an actuator disablement attack can overwrite the enablement action with a disablement action. By replacement, we mean that the attacker may have both enablement and disablement impacts on a controllable vulnerable event. Since we focus on attack prevention, by definition, research works that deal with replacement attacks also deal with enablement attacks as well as disablement attacks.

For this reason, we assign one of the dimensions of our framework to attack impact on the information sent through communication channels, which are essentially attack mechanisms on these channels.

6.2.3 Security Mechanism

The last dimension differentiating between the existing research is the defense mechanism that they provide. Generally, an attack is successful if it causes damage to the system or brings it to an undesirable mode. With the purpose of preventing failure of a supervisory control system under attack, two main approaches have been presented in the literature:

1. detection and prevention,
2. synthesis of a resilient supervisor.

The first approach places an intrusion detection module (IDM) in the system to detect an attack and prevent it before it causes damage to the system by leading it to an unsafe state [Carvalho et al. 2018; Lima et al. 2017]. The set of unsafe or critical states is assumed to be given, and they are mainly related to physical damage such as overflow or collision. As Figure 6.2 shows, the inserted module is connected to the supervisor and it observes the same events as the supervisor does. The module and the supervisor together will then form an integrated supervisor that determines control commands. To obtain such an IDM, first the behavior of the closed-loop control system under attack is modeled. The unsafe states of the closed-loop control system are those indicating an unsafe state of the plant. The module is implemented such that it detects an attack leading to an unsafe state of the attacked closed-loop control system. In this case, it will force the supervisor to take a proper action either by disabling all controllable events as proposed in Lima et al. [2017] or only those controllable events that can eventually cause the failure of the system as discussed in Lima et al. [2018].

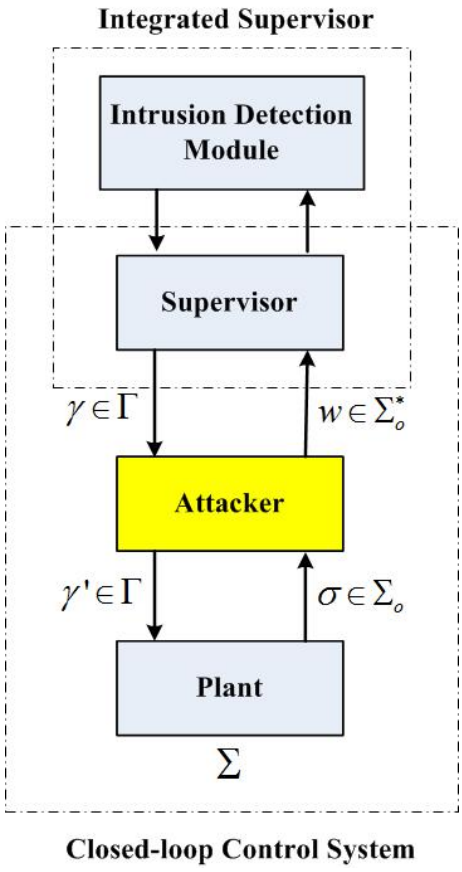


Figure 6.2: Intrusion detection module approach.

In the second approach, a supervisor is synthesized, which is resilient against attacks. Based on the synthesis method, this approach can be divided into the following main subcategories:

- direct synthesis,
- reducing the problem to conventional synthesis.

In the first method, a new synthesis algorithm is provided to obtain a resilient supervisor. In Su [2018], the author studies the problem of synthesis of resilient supervisors against sensor attacks. To solve the problem, first, a supremal successful attacker (an attacker that causes damage on the system) is synthesized, and based on that, a supervisor is synthesized that prevents system failure under attacks.

In the second method, conventional definitions of observability and controllability are first modified with respect to the effects of attacks on the system. Under the new conditions, the problem of supervisory control synthesis under attacks is reduced to the conventional synthesis problem [Wakaiki et al. 2017].

Both general security approaches and each synthesis method will be considered in the classification as follows:

1. IDM
2. resilient supervisor: $\begin{cases} \text{new synthesis} \\ \text{conventional synthesis} \end{cases}$

6.3 Classification

To compare the existing approaches investigating attack prevention, we first classify them based on the proposed framework. The classification is presented in Figure 6.3.

There are other important aspects that the classification can also focus on. However, due to the limited number of research works that are currently available, not all possibilities of these aspects have been studied in the literature. We have thus made the decision not to include them in the classification. Nevertheless, we discuss these important aspects in Section 6.6.

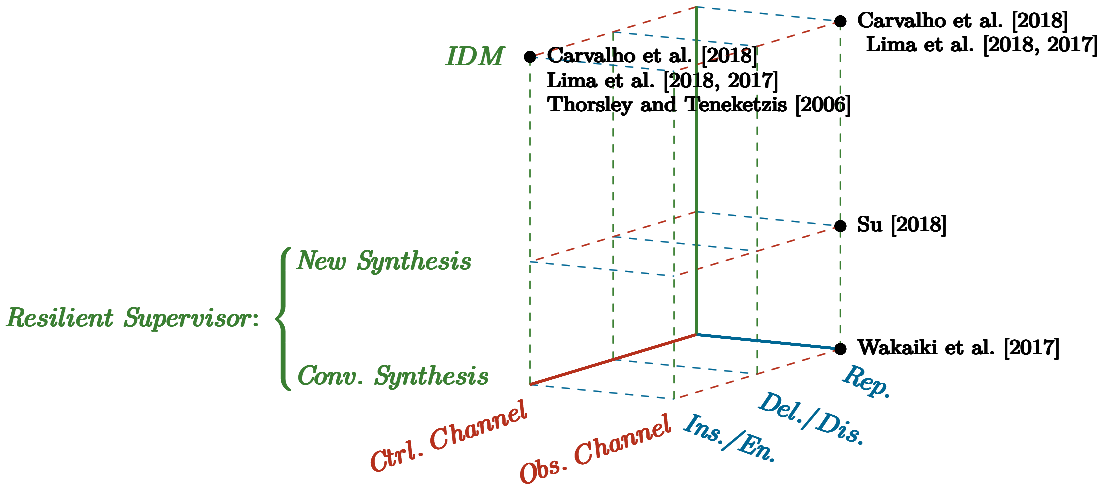


Figure 6.3: Classification of approaches on attack-prevention in supervisory control systems in the proposed framework.

6.4 Comparison of Security Approaches

In this section, we give an overview on advantages and disadvantages that each security approach brings. First, we focus on the two main approaches: IDM and resilient supervisory control synthesis. Then, we compare the two different directions that have been proposed for resilient supervisory control synthesis against attacks.

6.4.1 IDM vs Synthesis

The main difference between the existing IDM and resilient supervisor synthesis approach is that the first one critically relies on real-time attack detection and makes no assumption about the attacker, while the second approach constructs a supervisor so that the closed-loop system becomes non-attackable, by using a prior knowledge of the attack models. Each mechanism has its own advantages and limitations depending on the application and the type of attack as discussed in the following.

Attack model

As previously mentioned, in the IDM approach, no assumption is imposed on the attacker. The resilient synthesis approach imposes some assumptions on the attacker; in particular, it requires the attacker to know a model of the system. The assumption that the attacker knows the system model may seem a bit too strong. However, the assumption of a strong attacker is not necessarily bad as we focus on security and damage prevention. Indeed, in principle, if a resilient supervisor has been designed for an attacker that knows the system's model, then it will also be secure against attackers not having a system model. However, the existing resilient synthesis approach [Lin et al. 2018a; Su 2018], [Lin et al. 2018b] also assumes that the attacker tries to remain covert until causing guaranteed damage to the system. This additional assumption, which is not essential, limits the capability of the attacker (as no risk can be taken) and makes it harder to successfully attack the system. This limitation can be easily removed assuming a risky attacker, which we will discuss in Section 6.6.

Detectability

There are different types of attack strategies. An attack could be detectable if it changes the behavior of the system in such a way that the post-attack behavior is distinguishable from the normal behavior. However, there exist more complex types of attacks, which take the system model into consideration and use that knowledge to stay covert to the supervisor until the damage is caused. The implementation of an IDM is limited to systems satisfying the detectability condition, i.e., systems in which an attack can be detected before it causes any damage to the system by leading it to an unsafe state [Carvalho et al. 2018; Lima et al. 2017]. The detectability condition has been further modified in Lima et al. [2018] so that an attack is undetectable until reaching an unsafe boundary but may be detected after it reaches the boundary. While undetectable attacks cannot be handled by IDM, most of the resilient supervisor synthesis approaches focus on attacks that cannot be detected until causing damage [Góes et al. 2017; Su 2018; Wakaiki et al. 2017].

Closed-loop system behavior

As long as no attack has occurred, the IDM approach preserves the original closed-loop system behavior, while the resilient supervisor synthesis approach has to restrict the behavior of the closed-loop system from the beginning (in order to make it harder to attack), which limits the permissiveness of the supervisor. In the resilient supervisor synthesis approach, the supervisor can even be designed such that enablement attack may become impossible (in the actuator attack scenario) and thus normal execution may not have to be terminated[3] (depending on the permissiveness of the synthesized supervisor). The IDM approach has to terminate normal execution after detecting an attack. Compared with the resilient supervisor synthesis approach, instead of replacing all control logic implemented in the system, the IDM approach only needs to design and integrate an additional security module.

Defense philosophy

The defense philosophy in the IDM approach is quite straightforward. It relies solely on attack detection for preventing damage caused by attacks. This attack detection mechanism also exists in the synthesis approach, but a much richer defense philosophy can co-exist in the synthesis approach, due to the various assumptions imposed. For example, in the resilient synthesis approach, the attacker may be assumed to try to remain covert in its initial attacks before it can cause damage to the system in its final attack [Lin et al. 2018a; Su 2018]. This assumption on the attacker can be used to design supervisors that discourage the attacker from attacking the system. In particular, different assumptions on the attacker's model can lead to different defense strategies. These features do not exist in current implementations of the IDM approach.

6.4.2 New Synthesis vs. Conventional Synthesis

Here, we discuss the benefits and limitations of each method for resilient supervisory control synthesis, as far as the current research works are concerned. First of all, reducing the problem of supervisory control synthesis under attacks to conventional synthesis brings the advantage of being able to use the available techniques and tools. However, in Wakaiki et al. [2017], new conditions of controllability and observability are essentially proposed only for verification and thus they are quite restrictive. As presented in Su [2018], the direct synthesis approach involves a two-step synthesis since a supremal successful attacker needs to be synthesized first, which is then used to synthesize a resilient supervisor. In order to synthesize a supremal successful attacker, a normality assumption is imposed Lin et al. [2018a] and Su [2018], which says all vulnerable events are observable to the attacker and all controllable events are observable to the supervisor. This assumption is relaxed in Góes et al. [2017], and they construct an attacker allowing a larger class of attack strategies. Although the attacker is supposed to be at the basis to finding a resilient supervisor, this has not yet been provided in Góes et al. [2017] and Lin et al. [2018a]. Moreover, the

[3]For example, an extremely conservative approach is to treat each plant state with a vulnerable event defined as a bad state in the synthesis of the supervisor. Then, there is even no possibility of attack!

resilient supervisor synthesis algorithm provided in Su [2018] is sound but not complete. That is, it is possible that there exists a resilient supervisor but the algorithm in Su [2018] fails to find one.

6.5 Related Work

Besides papers studying security and resilience against attacks, there are works on robust supervisory control against problems with similar effects as attacks. These papers are briefly discussed here as they may contain ideas that can help us solve existing problems in the attack-prevention domain or inspire new research works. As we do not intend to conduct a comprehensive review of these related works, the reader is referred to the cited papers for more details.

6.5.1 Supervisory Control under Communication Problems

The effects of attacks could be the same as the effects of other communication problems on supervisory control systems. For instance, communication losses can be modeled and dealt with in the same way as deletion of vulnerable events sent through communication channels [Alves et al. 2014; Rohloff 2012; Ushio and Takai 2016; Yin 2017]. Moreover, the effects of communication delays can be modeled in the same way as deletion and insertion of vulnerable events. For instance, let us consider that an event, say $\sigma \in \Sigma_o$, is observed after a delay of one step (of the event occurrence), then in the delayed observed plant, σ is deleted at every state where it is enabled and inserted after the next transition. As already discussed in Chapter 3, there are several works investigating networked supervisory control of discrete-event systems with communication problems such as delays, losses and non-FIFO observations. These works include Lin [2014], the method presented in Chapter 3 and in Rashidinejad et al. [2018]. To deal with network communication delays and losses, in Lin [2014], a mapping is introduced to model the plant under observation delay and losses. To synthesize a networked supervisor, they provide new controllability and observability conditions under delay and losses called network controllability and network observability. For a plant and requirements satisfying network controllability and network observability conditions, a networked supervisor is obtained by applying the conventional synthesis.

Moreover, in Chapter 3 and in Rashidinejad et al. [2018], a non-FIFO observation channel is considered where events may be observed not necessarily in the same order as they have been executed in the plant. For instance, a word $w = \sigma_1\sigma_2$ where $\sigma_1, \sigma_2 \in \Sigma_o$ may be observed as $\sigma_1\sigma_2$ or $\sigma_2\sigma_1$. Non-FIFO observation has effects similar to replacement attacks in the sensor channel. In Chapter 3 and in Rashidinejad et al. [2018], a networked supervisor that is robust to communication delays and non-FIFO observations is synthesized through a new synthesis algorithm.

6.5.2 Fault-tolerant Supervisory Control of Discrete Event Systems

The impact of faults on supervisory control of a DES might be more general than deletion and insertion of vulnerable events in the attack domain. A fault may map the nominal behavior of the supervised system to a faulty behavior. However, fault-tolerant supervisory control approaches [Moor 2016] use quite similar mechanisms to deal with the effects of faults, which can be divided as:

- active fault tolerant supervisory control such as in Paoli and Lafortune [2005] and Paoli et al. [2011],
- passive fault tolerant supervisory control such as in Shu and Lin [2014] and Wittmann et al. [2012].

The first method is based on real-time fault detection and prevention. The strategy is the same as in the IDM approach. If a system satisfies diagnosability conditions, then a diagnoser is implemented. The diagnoser detects the occurrence of a fault before the system executes some illegal sequences. The nominal supervisor will then be switched to a new supervisor to satisfy some post-fault specifications [Paoli et al. 2011]. A fault diagnoser cannot be implemented for applications that do not satisfy diagnosability conditions.

The second approach provides a fault-tolerant supervisory control synthesis technique as discussed in Wittmann et al. [2012]. In this method, first, the nominal behavior of the plant and the faulty behavior are modeled as a fault-accommodating behavior. Using this model, an acceptable behavior is defined based on the desired nominal behavior and the acceptable defective behavior. As a result, the fault-tolerant supervisory control problem is transformed to a standard supervisory control problem under partial observations for which existing algorithms and tools can be used as discussed in Wittmann et al. [2012]. This approach is limited to faults occurring only once and with a certain degradation behavior.

6.5.3 Attacker Synthesis

Furthermore, there are a few works that only investigate attacker synthesis, as we have discussed before. For instance, in Góes et al. [2017] and Zhang et al. [2018], a successful insertion-deletion undetectable sensor attack is constructed and in Lin et al. [2018a], actuator attacks have been modeled under the assumption of a normality condition on the attacker and supervisor, in which case the supremal successful actuator attacker exists. In particular, under normality assumption on the supervisor, the attacker can exercise enablement attack for at most once. Moreover, under normality assumption on the attacker, disablement attack does not help enablement attack in achieving the attack goal, with the aim of remaining covert.

6.6 Topics to Investigate

All the discussed works follow the main objective of presenting a security approach that can handle most kinds of attacks, at least in principle. To achieve this goal, the current techniques still need to be improved in the following aspects.

6.6.1 Actuator Disablement Attack and Replacement Attack

As Figure 6.3 reveals, papers investigating synthesis of resilient supervisors have focused only on sensor attacks. [Lin et al. 2018a] provides an algorithm for the synthesis of a supremal successful actuator attacker under a normality assumption; however, no resilient supervisory control synthesis algorithm has been provided there. The problem has only been partially addressed in Zhu et al. [2019a] by solving an alternative problem of supervisor obfuscation. IDM approaches only considered enablement actuator attacks for the reason that disablement does not lead the system to an unsafe state, and it may only cause blocking issues (which are not important compared to security) [Carvalho et al. 2018]. On the other hand, it has been remarked in Lin et al. [2018a] that disablement attacks, in addition to enablement attacks, could be critical for a successful actuator attacker under the assumption that the attacker would like to remain covert. With a normality assumption imposed on the supervisor and attacker [Lin et al. 2018a], it has been shown that a replacement attacker is no more powerful than an enablement attacker. No synthesis algorithm has been provided for actuator replacement attacks for the general case without the normality assumption, and no resilient supervisor synthesis algorithm has been provided for the general case.

6.6.2 Attacks with Delay/Disordering Impact

All the available works assume that the attacker modifies the transmitted data instantaneously and does not change the relative ordering of data (events). However, an attacker may add delay in communication. It also may change the ordering of events sent through the same or different communication channels (either observation or control channels). Therefore, delaying and disordering of transmitted data can be other attack impacts besides insertion and deletion (or enablement and disablement) that still need to be investigated.

6.6.3 Non-risky versus risky attackers

A non-risky attacker carries out an attack only if 1) it remains covert after the attack, or 2) the attack will cause guaranteed damage to the system [Lin et al. 2018a; Su 2018]. On the other hand, being covert is typically not a goal of a risky attacker [Góes et al. 2017]. While most of the papers have focused on non-risky attackers, there could be scenarios in which an attacker is risky. In particular, a risky attacker implements a worst-case attack scenario and resilient supervisors against risky attackers tend to be less permissiveness. A risky sensor attacker has been synthesized in Góes et al. [2017]. However, there still does not exist any supervisor handling this type of attack.

6.6.4 Active Attacker with Partial Observation of Σ_o

An intelligent attacker (undetectable) can perform a successful attack based on its knowledge of the system model and its observation on the execution of the closed-loop system. However, an attacker may not always have full observation of Σ_o. In Lin et al. [2018a], a successful actuator attacker is synthesized that can only observe a subset of Σ_o, under the assumption that the attacker can observe the control command issued by the supervisor each time when the supervisor observes an event $\sigma \in \Sigma_o$. Based on the newly issued control command, the attacker can infer that some observable event of the supervisor has already been fired, even if that event is unobservable to the attacker. Moreover, the attacker and the supervisor satisfy a normality assumption. These conditions can be relaxed. Also, the supervisor may not issue a control command each time when it observes an execution of $\sigma \in \Sigma_o$, but instead it only sends the control command when it is necessary, which makes the closed-loop system harder to attack. However, no resilient supervisor synthesis algorithm is available under this condition.

6.6.5 Passive Attacks and Opacity Enforcement

As we have discussed, we have mainly focused on active attacks, that is, the attacker's goal is to inflict damage on the system. There is a more benign class of attacks, where the attacker's goal is to compromise the confidentiality of the system and learn secrets. An important notion for this purpose is opacity.

Opacity is a cyber-security property that recently has gained a lot of attention in supervisory control of DESs [Bérard et al. 2015; Dubreil et al. 2010; Jacob et al. 2016; Lafortune et al. 2018; Lin 2011; Saboori and Hadjicostis 2007, 2012; Wu and Lafortune 2014; Yin and Lafortune 2016; Yin and Li 2018b]. A system is opaque if an outside observer, who knows the model of the system but has only partial observation on the system evolution, cannot infer a "secret" about the system behavior [Jacob et al. 2016; Tong et al. 2018b]. In other words, the secret behavior is observationally equivalent to some non-secret behavior for the intruder. Opacity can be enforced either by an external enforcer or by synthesizing an opacity-enforcing supervisor as discussed in Cassez et al. [2012], Ji et al. [2018], Wu and Lafortune [2014], and Yin and Li [2018a] and in Tong et al. [2018b] and Yin and Lafortune [2016], respectively.

Consider an attacker that uses the knowledge of the system model to perform a successful covert attack, based on its partial observation on the execution of the system. If we make the unsafe behavior (strings from which the system can reach an unsafe state) opaque to the attacker, then the attacker may not be able to see an opportunity to make a successful attack. Securing a networked supervisory control system using opacity enforcement techniques is an ongoing research topic [Yin and Li 2018b].

6.6.6 Supervisor Obfuscation

As an alternative method for resilient supervisor synthesis, supervisor obfuscation can be used. For instance, in Zhu et al. [2019a], an algorithm is proposed to obfuscate a supervisor to make it resilient against actuator attacks, while preserving the behavior of the original

closed-loop system. An application of this approach is to first synthesize a supervisor that takes care of all the safety specifications, except for the resilience property; the supervisor can then be obfuscated to become resilient. This makes supervisor obfuscation another two-step synthesis method. However, a limitation of the algorithm in Zhu et al. [2019a] for solving the supervisor obfuscation problem is that it is sound but incomplete. Moreover, the efficiency of the proposed supervisor obfuscation algorithm in Zhu et al. [2019a] needs to be improved.

6.6.7 Other Topics

There are many other important topics that we are not able to cover here, including (probabilistic) attack on probabilistic systems, game based solving approach, model checking based solving approach, ideas using attack trees and graphs [Camtepe and Yener 2007; Jha et al. 2002; Kordy et al. 2014; Svoreňová and Kwiatkowska 2016] and so on. In addition, it is possible to investigate cooperative attack scenarios or decentralized attack architectures in similar spirit to [Tong et al. 2018a; Wu et al. 2018]. Last but not the least, it is important to consider the potential applications for securing practical industrial control systems. Towards this goal, a set of benchmark examples (see, for example, [Kang et al. 2016]) that mimic industrial control systems can be constructed for the researchers to test and compare the techniques and tools.

6.7 Conclusions

In this chapter, we have classified the current research works in the attack prevention of supervisory control systems using a framework consisting of the following dimensions: 1) the communication channel that is exposed to an attacker, 2) the attack impact on the transmitted data, and 3) the defense mechanism. We have also surveyed other relevant works investigating similar problems such as robust supervisory control approaches against faults, communication delays, losses and non-FIFO observations. Finally, we have pinpointed some under-explored topics for further investigations. Based on the discussions, we can conclude that this research topic is currently attracting a lot of attention from researchers in the supervisory control community, and many technical problems still need to be solved.

Chapter 7

Conclusion

7.1 Concluding Remarks

This thesis focuses on the supervisory control layer of a cyber-physical system (CPS). In a CPS, the communication between the physical world and the control unit is via a network. In general, controlling systems over a network is challenging due to communication delays, packet reordering, packet losses, and cyber-attacks that may be introduced by and through the network. These problems are neglected in theory while synthesizing a supervisor using the conventional supervisory control technique. Therefore, a conventional supervisor does not necessarily work when being implemented in a networked control setting, where the above mentioned problems may occur. This thesis aims to develop new supervisory control synthesis approaches, resulting in supervisors that handle the communication imperfections and the risk of cyber-attacks that may exist in practice.

Throughout this thesis, the research questions defined in Section 1.3 are answered as below.

> **Research Question 1:** *How can the supervisory control theory be improved in such a way that the synthesized supervisor copes with the problems that may appear in the PLC-implementation?*

In Chapter 2, an asynchronous supervisory control synthesis technique is presented in the framework of discrete-event systems (DESs) (see Section 2.3). It is assumed that the plant and the supervisor work in an asynchronous supervisory control setting, where the communication between those may not necessarily be synchronous. In addition, events executed in the plant in some order could be observed by the supervisor in a different order. Moreover, the plant can execute a controllable event only if it is commanded by the supervisor. Based on the asynchronous supervisory control setting, an algorithm is proposed that results in an asynchronous supervisor, guaranteeing that the asynchronously supervised plant behaves in the desired manner. As we already take care of the inexact synchronization, interleave sensitivity, and causality problems in the asynchronous supervisory control setting, we ensure that the asynchronous supervisor is

robust to those problems whenever they occur in the PLC-implementation.

> **Research Question 2:** *How can a supervisor be synthesized that is able to control a system over a communication network?*

Chapter 3 presents a networked supervisory control synthesis technique. For this purpose, a networked supervisory control (NSC) setting is provided in which the communication network may introduce delays and packet reordering. To model the effects of communication delays, it is necessary to have timing information of the event occurrences. In this respect, the plant is assumed to be given as a timed DES (TDES); a DES that incorporates discrete-time information by including a special event that indicates the passage of one unit of time. The behavior of the plant in the NSC setting is then derived as the networked plant. For the networked plant, an algorithm is proposed that results in a networked supervisor. It is formally proved that the networked supervisor guarantees controllability, nonblockingness, time-lock freeness, safety, and maximal permissiveness. As we already considered the communication delays and packet reordering in the NSC setting, we ensure that the networked supervisor is able to control the plant over a communication network as it handles the mentioned communication imperfections. The results of Chapter 3 are remarkable in the context of networked supervisory control firstly because incorporating timing information helps to provide a more precise model of the communication delays rather than measuring delays based on event occurrences, which is the case for most of the existing methods. In addition, our method guarantees to achieve the maximally permissive nonblocking networked supervisor, which according to [Lin 2020] is still an open issue in other techniques.

Using discrete time may introduce the state-space explosion problem for practical applications with large state spaces. To overcome this problem, Chapter 4 and Chapter 5 focus on timed automata (TA); a DES that incorporates dense-time information such that the state transitions do not only depend on the event occurrences but also on the time that events occur. Incorporating dense-time information not only helps to solve the state-space explosion problem but also provides a more natural way to model real-life applications compared to TDESs. In the literature of TA, many works appear with the purpose of verification, and there is not much work on supervisory control synthesis. Synthesizing a supervisor for a TA is challenging due to its infinite state space. To overcome this problem, the existing approaches first abstract time in a TA to achieve a finite automaton (FA) and then apply the SCT on the resulting FA. The synthesized supervisor is also an FA, and to translate it back into a TA, some time-refinement technique is required. The supervisor is allowed to control the plant only by strengthening the guards of edges labeled by controllable events, while the location invariants remain unchanged. Chapter 4 of this thesis improves the existing supervisory control synthesis approaches for TA in two aspects; first, it provides a less conservative result as the supervisor is allowed not only to strengthen the guards of edges labeled by controllable events but also the invariants of locations where forcible events occur. Second, it provides a time-refinement technique to achieve the supervisor also as a TA, which is not the case for most of the existing methods.

Further, in Chapter 5, we again use the concept of forcible events to provide a less conservative supervisor. However, this time we provide a synthesis algorithm that is directly applicable to a TA without being abstracted. The supervisory control synthesis approach presented in Chapter 5 is a remarkable achievement due to the following reasons:

- many practical applications for which it is necessary to guarantee safety and liveness are represented as TA rather than DESs,
- most of the available techniques and tools focus on verifying requirements for TA. However, our method achieves a supervisor that is correct-by-construction,
- the available synthesis techniques abstract TA into FA, which result in finite but very large state spaces for many practical applications. This problem is solved by our method as it involves no abstraction,
- we develop the conventional SCT presented by Ramadge and Wonham [1987] for TA rather than using game-based approaches. The supervisor obtained by game-based synthesis approaches does not necessarily preserve the largest admissible behavior of the plant. In Chapter 5, it is formally proved that besides controllability, nonblockingness, and safety, the synthesized supervisor guarantees maximal permissiveness of the supervised plant.

The result of Chapter 5 is a basis to develop networked supervisory control synthesis in a real-time framework.

> **Research Question 3:** *How can networked supervisory control be affected by cyber-attacks?*

Supervisory control synthesis under cyber-attacks is a relatively new emerging research topic. Chapter 6 provides an overview of attack prevention supervisory control approaches. A framework is presented to categorize existing methods. Based on this classification, missing pieces in this area are investigated. Chapter 6 also studies robust supervisory control approaches against faults, communication delays, packet reordering, and packet losses and discovers possible links between those and attack-prevention supervisory control techniques. The survey provided in Chapter 6 results in promoting new directions for further research in the area of supervisory control synthesis under cyber-attacks, which is growing rapidly.

7.2 Recommendations for Future Work

Synthesizing a supervisor for a CPS, modeled as a hybrid automaton (HA), which guarantees controllability, nonblockingness, safety, and maximal permissiveness requires further research and development. To move towards this goal, there are various ways to build further on the results of this research project. Below is a list of recommendations for future research.

7.2.1 Putting the Results into Practice

The synthesis algorithm presented in Chapter 5 facilitates the implementation as the proposed method works with automata instead of languages, and this eases the integration of an implementation in a toolset such as CIF [van Beek et al. 2014] or Supremica [Akesson et al. 2006]. Moreover, as the method involves no abstraction, it will be beneficial for practical applications with large state spaces.

7.2.2 Extensions Based on the Thesis Framework

Let us recall the framework discussed in Chapter 1. Figure 7.1 gives recommendations to extend the results of this thesis in the proposed directions. Details are given below.

Supervisory Control of HA

As stated in Cassandras and Lafortune [2009], a HA could be considered an extension of a TA, where clock dynamics are replaced by more complicated dynamical equations (time-driven dynamics). In this respect, the synthesis method presented in Chapter 5 is desired to be extended for HA. So far, to synthesize a supervisor for HA, existing methods first abstract the system to a DES (see [Koutsoukos et al. 2000]). Similar to TA, this abstraction may result in very large state spaces. Extending the method presented in Chapter 5 for HA can overcome this problem, and be beneficial for many practical applications.

Note that the semantic graph of a HA includes two types of transitions; transitions labeled by events, and transitions labeled by non-negative real values, which indicate the flow of the variables over time [Cuijpers et al. 2002; Henzinger 2000]. The non-negative real value transitions of a HA are dependent to a differentiable function and its first derivative. This dependency makes it much more challenging to compute the terms ③ and ⑥ in the nonblocking predicate (Algorithm 5.1) and the bad state predicate (Algorithm 5.2), respectively.

Supervisory Control of TA under Delays

The networked supervisory control technique proposed in Chapter 3 can be extended for TA. As a TA incorporates dense-time information, the networked supervisory control setting is modeled in a real-time framework. So, a more precise model of communication delays is achieved. For the real-time networked supervisory control setting, a networked plant can be derived also as a TA. To synthesize a supervisor for the networked plant, the method proposed in Chapter 5 can be used. As no abstraction is involved, it makes the method useful for many complex practical applications.

Supervisory Control of TDES under Attacks

The strategy of the networked supervisory control synthesis presented in Chapter 3 can be inspiring for attack-prevention supervisory control synthesis. Similarly, a supervisory control setting can be provided where the communication network may introduce some

types of cyber-attacks (as discussed in Chapter 6). Based on the proposed setting, the attacked plant can be obtained, modeling the behavior of the plant under the effects of cyber-attacks. Then, for the attacked plant, a supervisor can be synthesized using a similar synthesis algorithm presented in Chapter 3.

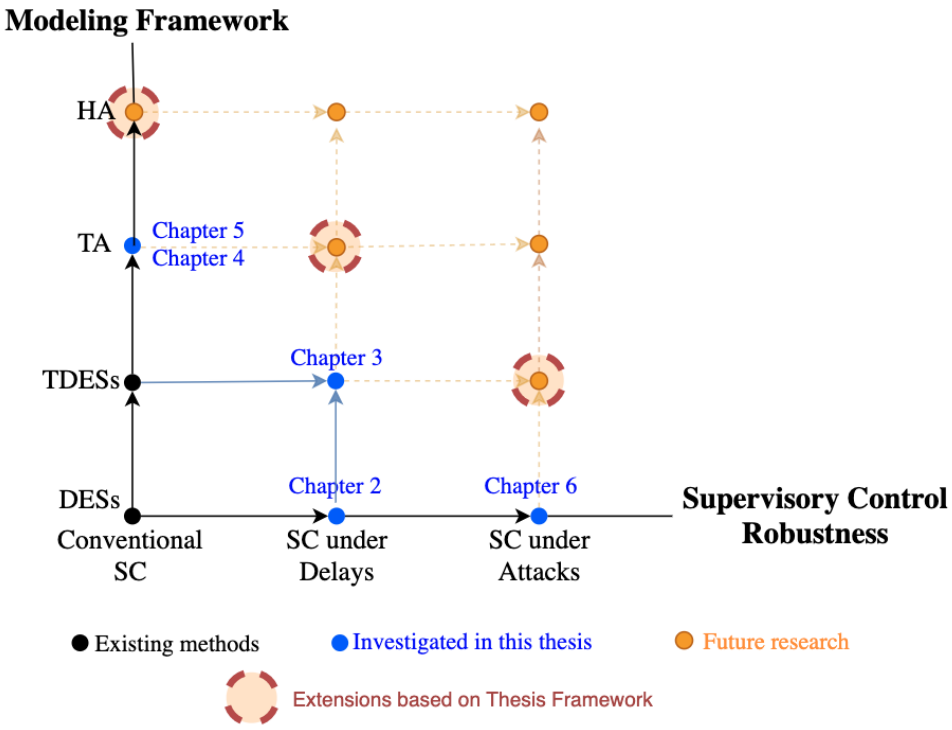


Figure 7.1: Extensions based on the proposed framework in Chapter 1.

7.2.3 Adding New Dimensions

In both the asynchronous supervisory control setting presented in Chapter 2 and the networked supervisory control setting presented in Chapter 3, all the plant events are assumed to be observable. In the case that a subset of the plant events is unobservable, the presented approaches should be extended by adapting the conventional definition of observability to the asynchronous/networked observability definition similar to what is presented in Lin [2014]. In this thesis, we focused on providing the maximally permissive supervisor, which according to [Lin 2020], is not the case for other existing networked supervisory control approaches even though they might consider partial observability. Extending the proposed approaches to asynchronous/networked supervisory control under partial observation is a further step of this research.

Appendix A

Proofs of Chapter 2

A.1 Technical Lemmas

Lemma A.1 (Nonblockingness over Projection)**.** If an automaton G with event set Σ and set of states A is nonblocking, then $P_{\Sigma'}(G)$ with event set $\Sigma' \subseteq \Sigma$ is also nonblocking.

Proof. Consider a set of reachable states A_r in $P_{\Sigma'}(G)$. Due to the definition of projection on automata, each state of $P_{\Sigma'}(G)$ is a subset of A, and so one can say $A_r \subseteq A$. By construction A_r is not empty and thus contains at least one element, say $a_r \in A$. By definition a_r is reachable in G and since G is nonblocking there is a $w \in \Sigma^*$ such that $\delta(a_r, w) \in A_m$. Then, again by construction and by the properties of the involved determinisation procedure (determinisation makes a state of $P_{\Sigma'}(G)$ marked if it includes at least one of the marked states of G [Hopcroft et al. 2006]) we have that $P_{\Sigma'}(w)$ allows to reach a marked state in $P_{\Sigma'}(G)$. ■

Lemma A.2 (Asynchronously Supervised Plant Transitions)**.** Considering the asynchronously supervised plant obtained from Definition 2.6, for any $w \in \Sigma^*_{ASP}$ with $\delta_{ASP}(z_0, w)!$, we have $\delta_{ASP}(z_0, w) = (\delta(a_0, P_\Sigma(w)), \delta_{AS}(y_0, P_{\Sigma_{AS}}(w)), m, l)$ for some $m \in M, l \in L$.

Proof. Let $w \in \Sigma^*_{ASP}$ with $\delta_{ASP}(z_0, w)!$. We prove that $\delta_{ASP}(z_0, w) = (\delta(a_0, P_\Sigma(w)),$ $\delta_{AS}(y_0, P_{\Sigma_{AS}}(w)), m, l)$ for some $m \in M, l \in L$ by induction on the structure of w. First, assume that $w = \epsilon$ then we have $\delta_{ASP}(z_0, w) = (a_0, y_0, [], \epsilon) = (\delta(a_0, \epsilon), \delta_{AS}(y_0, \epsilon), [], \epsilon)$. Now, assume that $w = v\sigma$ for some $v \in \Sigma^*_{ASP}$ with $\delta_{ASP}(z_0, v)!$. By induction we have $\delta_{ASP}(z_0, v) = (\delta(a_0, P_\Sigma(v)), \delta_{AS}(y_0, P_{\Sigma_{AS}}(v)), m', l')$ for some $m' \in M$ and $l' \in L$. It is sufficient to prove that $\delta_{ASP}(z_0, v\sigma) = (\delta(a_0, P_\Sigma(v\sigma)), \delta_{AS}(y_0, P_{\Sigma_{AS}}(v\sigma)), m, l)$ for some $m \in M$ and $l \in L$. For $\sigma \in \Sigma_{ASP}$ one of the following cases could occur; if $\sigma \in \Sigma$, then $\delta_{ASP}(z_0, v\sigma) = (\delta(\delta(a_0, P_\Sigma(v)), \sigma), \delta_{AS}(y_0, P_{\Sigma_{AS}}(v)), m, l)$, and if $\sigma \in \Sigma_{AS}$, then $\delta_{ASP}(z_0, v\sigma)$ $= (\delta(a_0, P_\Sigma(v)), \delta_{AS}(\delta_{AS}(y_0, P_{\Sigma_{AS}}(v)), \sigma), m, l)$ where in each case due to Definition 2.1 we get $\delta_{ASP}(z_0, v\sigma) = (\delta(a_0, P_\Sigma(v\sigma)), \delta_{AS}(y_0, P_{\Sigma_{AS}}(v\sigma)), m, l)$. ■

Lemma A.3 (Algorithm Termination)**.** The synthesis algorithm presented in Algorithm 2.1 terminates.

Proof. Let the output of the algorithm at iteration i be indicated by $AS(i)$. In each iteration, say iteration i, of Algorithm 2.1 at least one of its reachable states (from the nonempty set BS) is removed (by making all edges leading into those states undefined (Algorithm 2.1-line 7)). Since the automaton is finite state initially, this can only be done finitely often. ■

A.2 Proofs of Properties and Theorems

A.2.1 Proof of Property 2.1

Take $w \in L(AS|/|G)$ and $u \in \Sigma_{uc}$ such that $P_\Sigma(w)u \in L(G)$. Then, we need to prove that $wu \in L(AS|/|G)$. For $w \in L(AS|/|G)$, from Lemma A.2, we have $\delta_{ASP}(z_0, w) = (\delta(a_0, P_\Sigma(w)), \delta_{AS}(y_0, P_{\Sigma_{AS}}(w)), m, l)$ for some $m \in M, l \in L$. Then, from item 3 of Definition 2.6, we know that u occurs in $AS|/|G$ only if it is enabled by the plant where due to the assumption we have $\delta(a_0, P_\Sigma(w)u)!$. So from $\delta_{ASP}(z_0, w)$, the event u can occur which results in $\delta_{ASP}(z_0, wu) = (\delta(a_0, P_\Sigma(w)u), \delta_{AS}(y_0, P_{\Sigma_{AS}}(w)), m \uplus [u], l)$.

A.2.2 Proof of Property 2.2

$\Pi(G, N_c, N_o)$ is finite if it has a finite set of states and a finite set of events. Let us first prove that X is finite. Due to Definition 2.9, $X = A \times Q' \times M \times L$. To prove that X is a finite set, it is sufficient to guarantee that A, Q', M and L are finite sets because as proved in [Jech 2013] the Cartesian product of finite sets is finite. A is a finite set since we assumed that the plant is modeled by a finite automaton. For each $q' \in Q'$, we know that $q' \subseteq Q$. So, we should prove that Q is a finite set. $Q = A \times M$ is finite since A and M are finite as the maximum size of M is limited to a finite number N_o. Finally, L is finite since its size is limited to N_c.

A.2.3 Proof of Theorem 2.1

$AS|/|G = (Z, \Sigma_{ASP}, \delta_{ASP}, z_0, Z_m)$ is nonblocking if for all $z \in Reach(z_0)$ there exists a word $w \in \Sigma^*_{ASP}$ such that $\delta_{ASP}(z, w) \in Z_m$. Take $z \in Reach(z_0)$ where $z = (a, y, m, l)$, then we need to find $w \in \Sigma^*_{ASP}$ for which $\delta(a, P_\Sigma(w)) \in A_m$ and $\delta_{AS}(y, P_{\Sigma_{AS}}(w)) \in Y_m$ since $Z_m = A_m \times Y_m \times M \times L$. From Lemma A.2, we know that $\delta_{ASP}(z_0, w) = (\delta(a_0, P_\Sigma(w)), \delta_{AS}(y_0, P_{\Sigma_{AS}}(w)), m, l)$ for some $m \in M, l \in L$. We also have AS is nonblocking when Algorithm 2.1 terminates successfully which means that there is no blocking state to be removed ($BS = \varnothing$) and also due to Property A.1 the projection operator does not change the nonblockingness of an automaton. So, we can say that for $y \in Reach(y_0)$ there exists a word $w' \in \Sigma^*_{AS}$ such that $\delta_{AS}(y, w') \in Y_m$. From Algorithm 2.1, we have $AS \subseteq P_{\Sigma_{AS}}(AP)$ since we start the algorithm from $AS = AP$ and we remove transitions leading to blocking states, and finally we use the projection to leave out the Σ events. The state $y \subseteq X$ is a set of observationally equivalent states of AP. Since $\delta_{AS}(y, w')!$, we can say

that $\forall x \in y, \exists w \in \Sigma^*_{ASP}, \delta_{AP}(x,w) \in X_m$ because otherwise all observationally equivalent transitions have been removed and $\neg\delta_{AS}(y,w')$!. Take $w \in \Sigma^*_{ASP}, \delta_{AP}(x,w) \in X_m$, we only need to prove that $\delta(a, P_\Sigma(w)) \in A_m$. $\delta_{AP}(x,w) \in X_m, X_m = A_m \times Q' \times M \times L$. So, $\delta_{AP}(x,w) \in X_m$ implies that $\delta(a, P_\Sigma(w)) \in A_m$.

Appendix B

Proofs of Chapter 3

B.1 Technical Lemmas

Here, the notation . is used to refer to an element of a tuple. For instance, $z.a$ refers to the (first) element a of $z = (a, y, m, l)$.

Lemma B.1 (Nonblockingness over Projection)**.** For any TDES G with event set Σ and any event set $\Sigma' \subseteq \Sigma$: if G is nonblocking, then $P_{\Sigma'}(G)$ is nonblocking.

Proof. Consider an arbitrary TDES $G = (A, \Sigma, \delta, a_0, A_m)$ and arbitrary $\Sigma' \subseteq \Sigma$. Suppose that G is nonblocking. Consider an arbitrary reachable state $A_r \subseteq A$ in $P_{\Sigma'}(G)$. By construction A_r is nonempty. Assume that this state is reached through the word $w \in \Sigma'$. Then, for each state $a \in A_r$, again by construction, $\delta(a_0, w') = a$ for some $w' \in \Sigma^*$ with $P_{\Sigma'}(w') = w$. Because G is nonblocking, there exists a $v' \in \Sigma^*$ such that $\delta(a, v') = a_m$ for some $a_m \in A_m$. Consequently, from state A_r, it is possible to have a transition labelled with $P_{\Sigma'}(v')$ to a state A'_r containing a_m. By construction, this state A'_r is a marked state in $P_{\Sigma'}(G)$. Hence, the projection automaton is nonblocking as well. ■

Lemma B.2 (Time-lock Freeness over Projection)**.** For any TDES G with event set Σ and any event set $\Sigma' \subseteq \Sigma$, *tick* $\in \Sigma'$: if G is TLF , then $P_{\Sigma'}(G)$ is TLF.

Proof. The proof is similar to the proof of Lemma B.1. ■

Lemma B.3 (*NSP* Transitions)**.** Given a plant G, networked supervisor *NS* (for that plant) and networked supervised plant *NSP* (for those): $\delta_{NSP}(z_0, w).a = \delta_G(a_0, P_{\Sigma_G}(w))$ and $\delta_{NSP}(z_0, w).y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(w))$, for any $w \in L(NSP)$.

Proof. Take $w \in L(NSP)$, we prove that $\delta_{NSP}(z_0, w).a = \delta_G(a_0, P_{\Sigma_G}(w))$ and $\delta_{NSP}(z_0, w).y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(w))$ by induction on the structure of w.

Base case: Assume that $w = \epsilon$. Then, $\delta_{NSP}(z_0, w).a = a_0 = \delta_G(a_0, P_{\Sigma_G}(\epsilon))$ and $\delta_{NSP}(z_0, w).y = y_0 = \delta_{NS}(y_0, P_{\Sigma_{NS}}(\epsilon))$.

Induction step: Assume that $w = v\sigma$, where the statement holds for v, meaning that $\delta_{NSP}(z_0, v).a = \delta_G(a_0, P_{\Sigma_G}(v))$ and $\delta_{NSP}(z_0, v).y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v))$. It suffices to prove that the statement holds for $v\sigma$, i.e., $\delta_{NSP}(z_0, v\sigma).a = \delta_G(a_0, P_{\Sigma_G}(v\sigma))$ and $\delta_{NSP}(z_0, v\sigma).y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v\sigma))$. Considering Definition 3.7, for σ enabled at $\delta_{NSP}(z_0, v)$ the following cases may occur:

$\sigma \in \Sigma_{NS} \setminus \{tick\}$, which refers to item 1) and item 5). Then, $\delta_{NSP}(z_0, v).a$ remains unchanged; $\delta_{NSP}(z_0, v\sigma).a = \delta_{NSP}(z_0, v).a = \delta_G(a_0, P_{\Sigma_G}(v)) = \delta_G(a_0, P_{\Sigma_G}(v)\epsilon) = \delta_G(a_0, P_{\Sigma_G}(v\sigma))$, and $\delta_{NSP}(z_0, v\sigma).y = \delta_{NS}(\delta_{NS}(y_0, P_{\Sigma_{NS}}(v), \sigma) = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v)\sigma) = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v\sigma))$.

$\sigma \in \Sigma_G \setminus \{tick\}$, which refers to item 2) and item 3). Then, $\delta_{NSP}(z_0, v\sigma).a = \delta_G(\delta_G(a_0, P_{\Sigma_G}(v)), \sigma) = \delta_G(a_0, P_{\Sigma_G}(v)\sigma) = \delta_G(a_0, P_{\Sigma_G}(v\sigma))$, and $\delta_{NSP}(z_0, v\sigma).y$ remains unchanged such that $\delta_{NSP}(z_0, v\sigma).y = \delta_{NSP}(z_0, v).y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v)\epsilon) = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v\sigma))$.

$\sigma = tick$, which refers to item 4). Then, $\delta_{NSP}(z_0, v\sigma).a = \delta_G(\delta_G(a_0, P_{\Sigma_G}(v)), \sigma) = \delta_G(a_0, P_{\Sigma_G}(v)\sigma) = \delta_G(a_0, P_{\Sigma_G}(v\sigma))$. Also, $\delta_{NSP}(z_0, v\sigma).y = \delta_{NS}(\delta_{NS}(y_0, P_{\Sigma_{NS}}(v)), \sigma) = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v)\sigma) = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v\sigma))$.

Conclusion: By the principle of induction, the statement ($\delta_{NSP}(z_0, w).a = \delta_G(a_0, P_{\Sigma_G}(w))$ and $\delta_{NSP}(z_0, w).y = \delta_{NS}(v_0, P_{\Sigma_{NS}}(w))$) holds for all $w \in L(NSP)$. ■

Lemma B.4 (NP Transitions)**.** Given a plant G with $x_0.a = a_0$, for any $w \in L(NP)$: $\delta_{NP}(x_0, w).a = \delta_G(a_0, P_{\Sigma_G}(w))$.

Proof. The proof is similar to the proof of Lemma B.3. ■

Lemma B.5 (NP Enabling Commands)**.** Given a plant G, events from Σ_e are enabled on time in the networked plant NP (for that plant); for any $w\sigma \in L(G)$, $\sigma \in \Sigma_c$: there exists $w_0\sigma_e w_1\sigma \in L(NP)$ where $\sigma_e \in \Sigma_e$ is the enabling event of σ, $P_{\Sigma_G}(w_0w_1) = w$ and $|P_{\{tick\}}(w_1)| = N_c$.

Proof. Assume that $G' = P_{\Sigma_G \setminus \Sigma_{uc}}(G)$ is represented by $(A', \Sigma_G, \delta'_G, a'_0, A'_m)$, and let us do the proof by induction on the number of controllable events in $w \in L(G)$.

Base case: Assume that σ is the 1^{th} controllable event enabled in G. Then, $w \in (\Sigma_{uc} \cup \{tick\})^*$. According to Assumption 1, $|P_{\{tick\}}(w)| \geq N_c$. Let say $|P_{\{tick\}}(w)| = N_c + i$ for some $i \in \mathbb{N}_0$. Then, $tick^{N_c+i}\sigma \in L(G')$. Also, assume that $w = w_i w_{N_c}$ for some w_i, w_{N_c} where $|P_{\{tick\}}(w_i)| = i$, and $|P_{\{tick\}}(w_{N_c})| = N_c$. Considering Definition 3.10, NP starts from $x_0 = (a_0, \delta'_G(a'_0, tick^{N_c}), [], \epsilon)$. Then, based on item 4), *tick* occurs in NP when it is enabled in both G and G'. For the first i *ticks*, whenever *tick* is enabled in G, it is also enabled in G' (there are i *ticks* enabled in G' before σ occurs). Meanwhile, if there is an event ready to be observed, then based on item 5), the corresponding observed event occurs in NP which does not change the current state of G and G'. Also, based on item 3), if an uncontrollable event is enabled in G, it occurs in NP without changing the state of G'. Otherwise, *tick* occurs in NP by being executed in both G and G'. We call this situation as G and G' are synchronized on *tick*. Therefore, it is feasible that some w_0 is executed in NP based on the execution of $tick^i$ in G' and w_i in G. Then, $\delta_{NP}(x_0, w_0).a = \delta_G(a_0, w_i)$ and $\delta_{NP}(x_0, w_0).a' = \delta'_G(a_0, tick^{N_c+i})$, and so $P_{\Sigma_G}(w_0) = w_i$.

After that, since $(\delta_{NP}(x_0, w_0).a', \sigma)!$, based on item 1), σ_e occurs in NP, and (σ, N_c) is added to $\delta_{NP}(x_0, w_0).l$. Note that based on item 3) (item 5)), uncontrollable events enabled in G (events ready to be observed) can occur in between, but without loss of generality, let us assume that σ_e is enabled first, and then uncontrollable (observed) events are executed. So, w_1 will be executed in NP based on the execution of w_{N_c} in G. Therefore, $P_{\Sigma_G}(w_1) = w_{N_c}$, and $|P_{\{tick\}}(w_1)| = |P_{\{tick\}}(w_{N_c})| = N_c$. Based on item 4), by the execution of each *tick*, $\delta_{NP}(x_0, w_0\sigma_e).l$ is decreased by one. Also, σ is the only controllable event enabled in G so that $head(\delta_{NP}(x_0, w_0\sigma_e w_1).l) = (\sigma, 0)$. Then, based on item 2), σ will be executed in NP.

Induction step: Assume that σ is the n^{th} controllable event enabled in G where the statement holds for all previous controllable events. Let us indicate the $(n-1)^{th}$ controllable event by σ^{n-1} such that $w_{n-1}\,\sigma^{n-1} \in L(G)$. As the statement holds for σ^{n-1}, there exists some $w_0^{n-1}\,\sigma_e^{n-1}\,w_1^{n-1}\,\sigma^{n-1} \in L(NP)$ such that $P_{\Sigma_G}(w_0^{n-1}\,w_1^{n-1}) = w_{n-1}$ and $|P_{\{tick\}}(w_1^{n-1})| = N_c$. It suffices to prove that for the next controllable event σ^n, $w_n\,\sigma^n \in L(G)$, there exists some $w_0^n\,\sigma_e\,w_1^n\,\sigma^n \in L(NP)$ with $P_{\Sigma_G}(w_0^n\,w_1^n) = w_n$ and $|P_{\{tick\}}(w_1^n)| = N_c$. Let us say $w_n = w_{n-1}\,\sigma^{n-1}w$ where $w \in (\Sigma_{uc} \cup \{tick\})^*$. Assume that $|P_{\{tick\}}(w)| = j$, and $w_{n-1} = w_i^{n-1}\,w_{N_c}^{n-1}$ where $|P_{\{tick\}}(w_i^{n-1})| = i$, and $|P_{\{tick\}}(w_{N_c}^{n-1})| = N_c$. Moreover, let us say for $w_{n-1}\,\sigma^{n-1}\,w\,\sigma \in L(G)$, there exists $tick^{N_c}\,w'_{n-1}\,\sigma^{n-1}\,tick^j\sigma^n \in L(G')$ where $|P_{\{tick\}}(w'_{n-1})| = i$. Considering Definition 3.10, G' synchronizes with G on executing *tick* since whenever *tick* is enabled in G', it occurs in NP only if G enables it as well. Also, uncontrollable events (observed events) occur as they are enabled in G (as the corresponding event is ready to be observed), and due to the induction assumption, all controllable events occurring in G before σ^n are enabled on time, and so they will be executed in NP. By the execution of $w_0^{n-1}\,\sigma_e^{n-1}$ in NP, $\delta_{NP}(x_0, w_0^{n-1}\,\sigma_e^{n-1}).a' = \delta'_G(a'_0, tick^{N_c}\,w'_{n-1}\,\sigma^{n-1})$, and so $\delta_{NP}(x_0, w_0^{n-1}\,\sigma_e^{n-1}).a = \delta_G(a_0, w_i^{n-1})$ (G and G' synchronize on *tick*). At this point, (before reaching σ) in G', $tick^j$ is enabled, and $w_{N_C}^{n-1}$ is enabled in G (before reaching σ^{n-1}). Then, one of the following cases may occur:

$j < N_c$. Then, assume $w_{N_c}^{n-1} = w_j^{n-1}w_{N_c-j}^{n-1}$ for some $w_j^{n-1}, w_{N_c-j}^{n-1}$, where $|P_{\{tick\}}w_j^{n-1}| = j$ and $|P_{\{tick\}}w_{N_c-j}^{n-1}| = N_c - j$. Then, the execution of $tick^j$ in G' is synchronized with the execution of w_j^{n-1} in G resulting in $w_0^{n-1}\sigma_e^{n-1}v_1 \in L(NP)$ with $|P_{\{tick\}}(v_1)| = j$ and $P_{\Sigma_G}(v_1) = w_j^{n-1}$. After that σ_e occurs in NP (as it is enabled in G') adding (σ, N_c) to l. This follows by the execution of $w_{N_c-j}^{n-1}$ in G and results in $w_0^{n-1}\sigma_e^{n-1}v_1\sigma_e^n v_2 \in L(NP)$ where $|P_{\{tick\}}(v_2)| = N_c - j$ and $P_{\Sigma_G}(v_2) = w_{N_c-j}^{n-1}$. At this point, σ^{n-1} is executed in NP following by the execution of v_3 where $P_{\Sigma_G}(v_3) = w$, and so $|P_{\{tick\}}(v_3)| = j$. This results in $w_0^{n-1}\,\sigma_e^{n-1}\,v_1\,\sigma_e^n\,v_2\,\sigma^{n-1}\,v_3 \in L(NP)$ where $P_{\{tick\}}(v_2\sigma^{n-1}v_3) = N_c - j + j = N_c$, and $head(l) = (\sigma^n, 0)$ (after the execution of σ^{n-1}, this is only (σ^n, N_c) in l, and l is decreased by one by the execution of each *tick*), and so σ^n occurs in NP. Hence, $w_0^n\,\sigma_e^n\,w_1^n\,\sigma^n \in L(NP)$ for $w_0^n = w_0^{n-1}\,\sigma_e^{n-1}\,v_1$ and $w_1^n = v_2\,\sigma^{n-1}\,v_3$ where $P_{\Sigma_G}(w_0^n\,w_1^n) = P_{\Sigma_G}(w_0^{n-1}\,\sigma_e^{n-1}\,v_1\,v_2\,\sigma^{n-1}\,v_3) = w^n$ and $|P_{\{tick\}}(w_1^n)| = N_c - j + j = N_c$.

$j > N_c$. Then, assume $w = w_{j-N_c}w_{N_c}$ for some w_{j-N_c}, w_{N_c}, where $|P_{\{tick\}}w_{j-N_c}| = j - N_c$ and $|P_{\{tick\}}w_{N_c}| = N_c$. The execution of $tick^{N_c}$ in G' is synchronized with the execution of $w_{N_c}^{n-1}$ in G resulting in $w_0^{n-1}\,\sigma_e^{n-1}\,w_1^{n-1}\,\sigma_{n-1} \in L(NP)$. After that, the execution of the remaining $tick^{j-N_c}$ in G' will be synchronized with execution of w_{j-N_c}

in G resulting in $w_0^{n-1}\,\sigma_e^{n-1}\,w_1^{n-1}\,\sigma^{n-1}\,v_1 \in L(NP)$ where $P_{\Sigma_G}(v_1) = w_{j-N_c}$. Then, σ_e^n occurs in NP as it is enabled in G' adding (σ^n, N_c) to l. Finally, the execution of w_{N_c} in G results in $w_0^{n-1}\sigma_e^{n-1}w_1^{n-1}\sigma^{n-1}v_1\sigma_e^n v_2 \in L(NP)$ with $P_{\Sigma_G}(v_2) = w_{N_c}$. As N_c *ticks* have passed, $head(l) = (\sigma^n, 0)$, and σ^n occurs in NP. Hence, $w_0^n\,\sigma_e^n\,w_1^n\,\sigma^n \in L(NP)$ for $w_0^n = w_0^{n-1}\,\sigma_e^{n-1}\,w_1^{n-1}\,\sigma^{n-1}\,v_1$ and $w_1^n = v_2$ where $P_{\Sigma_G}(w_0^n\,w_1^n) = w_n$ and $|P_{\{tick\}}(w_1^n)| = N_c$.

$j = N_c$. Then, after the execution of $w_0^{n-1}\sigma_e^{n-1}$ in NP, v_1 occurs in NP related to the execution of $w_{N_c}^{n-1}$ in G and w_n' in G'. At this point, σ^{n-1} is enabled in G and $head(l) = (\sigma^{n-1})$. Also, σ^n is enabled in G'. Therefore, either $\sigma_e^n\sigma^{n-1}$ or $\sigma^{n-1}\sigma_e^n$ occurs in NP both followed by the execution of some v_2 in NP such that $P_{\Sigma_G}(v_2) = w$. As N_c *ticks* have passed ($head(l) = (\sigma^n, 0)$), and σ^n is enabled in G, σ^n occurs in NP. This results in one of the following words; $w_0^{n-1}\sigma_e^{n-1}v_1\sigma^{n-1}\sigma_e^n v_2\sigma^n \in L(NP)$ or $w_0^{n-1}\sigma_e^{n-1}v_1\sigma_e^n\sigma^{n-1}v_2\sigma^n \in L(NP)$ where the statement holds in both cases as already discussed in the previous items.

Conclusion: By the principle of induction, the statement holds for all $\sigma \in \Sigma_c$ and $w \in \Sigma_G^*$ with $w\sigma \in L(G)$. ■

Lemma B.6 (NSP and NP)**.** Consider a plant G, a networked supervisor NS (for that plant), the observation channel M, and the control channel L. The networked plant NP has the set of states X and the networked supervised plant NSP has the set of states Z. Then, for any pair of $x \in X$ and $z \in Z$ reachable through the same $w \in \Sigma_{NSP}^*$: $x.m = z.m$ and $x.l = z.l$.

Proof. Take $x \in X$, $z \in Z$, and $w \in \Sigma_{NSP}^*$ such that $x = \delta_{NP}(x_0, w)$ and $z = \delta_{NSP}(z_0, w)$. By induction on the structure of w, it is proved that $x.m = z.m$ and $x.l = z.l$.

Base case: Assume $w = \epsilon$. Then, $x.m = x_0.m = []$ ($x.l = x_0.l = \epsilon$), and $z.m = z_0.m = []$ ($z.l = z_0.l = \epsilon$). Thereto, $x.m = z.m$ and $x.l = z.l$.

Induction step: Assume $w = v\sigma$ where the statement holds for $v \in \Sigma_{NSP}^*$ and the intermediate states reached by v so that $\delta_{NP}(x_0, v).m = \delta_{NSP}(z_0, v).m$ and $\delta_{NP}(x_0, v).l = \delta_{NSP}(z_0, v).l$. It suffices to prove that the statement holds for $v\sigma$, i.e., $\delta_{NP}(x_0, v\sigma).m = \delta_{NSP}(z_0, v\sigma).m$ and $\delta_{NP}(x_0, v\sigma).l = \delta_{NSP}(z_0, v\sigma).l$. Considering Definition 3.10 and Definition 3.7, in both operators, $\delta_{NP}(x_0, v).m$ ($\delta_{NSP}(z_0, v).m$) changes by the execution of $\sigma \in \Sigma_c \cup \Sigma_{uc} \cup tick \cup \Sigma_o$ (item 2), item 3), item 4), and item 5)), and $\delta_{NP}(x_0, v).l$ ($\delta_{NSP}(z_0, v).l$) changes by the execution of $\sigma \in \Sigma_e \cup \Sigma_c \cup tick$ (item 1), item 2), and item 4)) in a similar way. Therefore, starting from $\delta_{NP}(x_0, v)$ and $\delta_{NSP}(z_0, v)$ with $\delta_{NP}(x_0, v).m = \delta_{NSP}(z_0, v).m$ ($\delta_{NP}(x_0, v).l = \delta_{NSP}(z_0, v).l$), the execution of the same event σ results in $\delta_{NP}(x_0, v\sigma).m = \delta_{NSP}(z_0, v).m$ ($\delta_{NP}(x_0, v\sigma).l = \delta_{NSP}(z_0, v).l$).

Conclusion: By the principle of induction, the statement ($x.m = z.m$ and $x.l = z.l$) holds for all $w \in \Sigma_{NSP}^*$, $x = \delta_{NP}(x_0, w)$ and $z = \delta_{NSP}(z_0, w)$. ■

Lemma B.7 (*NSP* and Product)**.** Given a plant G and a networked supervisor NS with event set Σ_{NS}. If $L(NS) \subseteq P_{\Sigma_{NS}}(L(NP))$, then $L(NSP) = L(NS||NP)$.

Proof. This is proved in two steps; 1. for any $w \in L(NSP)$: $w \in L(NS||NP)$, and 2. for any $w \in L(NS||NP)$: $w \in L(NSP)$.

1) Take $w \in L(NSP)$. By induction on the structure of w, it is proved that $w \in L(NS||NP)$.

Base case: Assume that $w = \epsilon$. Then, $w \in L(NS||NP)$ by definition.

Induction step: Let $w = v\sigma$ for some $v \in \Sigma^*_{NSP}$ and $\sigma \in \Sigma_{NSP}$ where the statement holds for v, i.e., $v \in L(NS||NP)$. It suffices to prove that the statement holds for $v\sigma$, i.e., $v\sigma \in L(NS||NP)$. Due to Lemma B.3, $\delta_{NSP}(z_0, v).y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v))$, $\delta_{NSP}(z_0, v\sigma).y = \delta_{NS}(\delta_{NSP}(z_0, v).y, P_{\Sigma_{NS}}(\sigma))$, $\delta_{NSP}(z_0, v).a = \delta_G(a_0, P_{\Sigma_G}(v))$, and $\delta_{NSP}(z_0, v\sigma).a = \delta_G(\delta_{NSP}(z_0, v).a, P_{\Sigma_G}(\sigma))$. Due to the definition of synchronous product (in [Cassandras and Lafortune 2009]), since $\Sigma_{NS} \subseteq \Sigma_{NSP}$, one can say any $w \in L(NS||NP)$ if $w \in L(NP)$ and $P_{\Sigma_{NS}}(w) \in L(NS)$. For $v\sigma \in L(NSP)$, it is already showed that $P_{\Sigma_{NS}}(v\sigma) \in L(NS)$, and so it suffices to prove $v\sigma \in L(NP)$. For $v \in L(NS||NP)$: $v \in L(NP)$ (since $\Sigma_{NS} \subseteq \Sigma_{NSP}$). Then, due to Lemma B.4, $\delta_{NSP}(z_0, v).a = \delta_G(a_0, P_{\Sigma_G}(v)) = \delta_{NP}(x_0, v).a$. Also, both $\delta_{NSP}(z_0, v)$ and $\delta_{NP}(x_0, v)$ are reachable through v, and so due to Lemma B.6, $\delta_{NSP}(z_0, v).m = \delta_{NP}(x_0, v).m$ and $\delta_{NSP}(z_0, v).l = \delta_{NP}(x_0, v).l$. Since $P_{\Sigma_{NS}}(v\sigma) \in L(NS)$ (for $v\sigma \in L(NSP)$, $\delta_{NS}(y_0, v\sigma)!$ due to Lemma B.3), and $L(NS) \subseteq P_{\Sigma_{NS}}(L(NP))$, then one can say there exists $w' \in L(NP)$, $P_{\Sigma_{NS}}(w') = P_{\Sigma_{NS}}(v\sigma)$. Without loss of generality, assume $w' = v' P_{\Sigma_{NS}}(\sigma)$ where $P_{\Sigma_{NS}}(v') = P_{\Sigma_{NS}}(v)$. Let us complete the proof for different cases of $\sigma \in \Sigma_{NSP}$.

$\sigma \in \Sigma_e$. Then, $\delta'_G(\delta_{NP}(x_0, v).a', \sigma)!$ since $\delta_{NP}(x_0, v).a' = \delta_{NP}(x_0, v').a'$ and $\delta'_G(\delta_{NP}(x_0, v').a', \sigma)!$ ($P_{\Sigma_e \cup \{tick\}}(v) = P_{\Sigma_e \cup \{tick\}}(v')$, and due to Definition 3.10, $x.a'$ changes by $w \in (\Sigma_e \cup \{tick\})^*$). So, due to item 1), $\delta_{NP}(\delta_{NP}(x_0, v), \sigma)!$.

$\sigma \in \Sigma_c$. Then, $\delta_G(\delta_{NP}(x_0, v).a, \sigma)!$ for the reason that $\delta_{NP}(x_0, v).a = \delta_{NSP}(z_0, v).a$ and $\delta_G(\delta_{NSP}(z_0, v).a, \sigma)!$. Also, the condition $head(\delta_{NP}(x_0, v).l) = (\sigma, 0)$ is satisfied since $\delta_{NP}(x_0, v).l = \delta_{NSP}(z_0, v).l$ and $head(\delta_{NSP}(z_0, v).l) = (\sigma, 0)$ (considering Definition 3.7-item 2)), σ can occur only if $head(\delta_{NSP}(z_0, v).l) = (\sigma, 0))$. So, due to Definition 3.10-item 2), $\delta_{NP}(\delta_{NSP}(z_0, v), \sigma)!$.

$\sigma \in \Sigma_{uc}$. Then, $\delta_G(\delta_{NP}(x_0, v).a, \sigma)!$ for the reason that $\delta_{NP}(x_0, v).a = \delta_{NSP}(z_0, v).a$ and $\delta_G(\delta_{NSP}(z_0, v).a, \sigma)!$. So, based on Definition 3.10-item 3), $\delta_{NP}(\delta_{NP}(x_0, v), \sigma)!$.

$\sigma = tick$. Then, $\delta_G(\delta_{NP}(x_0, v).a, \sigma)!$ for the reason that $\delta_{NP}(x_0, v).a = \delta_{NSP}(z_0, v).a$ and $\delta_G(\delta_{NSP}(z_0, v).a, \sigma)!$. Also, $\delta'_G(\delta_{NP}(x_0, v).a', \sigma)!$ since $\delta_{NP}(x_0, v).a' = \delta_{NP}(x_0, v').a'$ and $\delta'_G(\delta_{NP}(x_0, v').a', \sigma)!$. Moreover, $(\sigma, 0) \notin \delta_{NP}(x_0, v).m$ for all $\sigma \in \Sigma_a$ since $\delta_{NSP}(z_0, v).m = \delta_{NP}(x_0, v).m$ and $(\sigma, 0) \notin \delta_{NSP}(z_0, v).m$ (considering Definition 3.7-item 4), *tick* can occur if $(\sigma, 0) \notin \delta_{NSP}(z_0, v).m$). Therefore, based on Definition 3.10-item 4), $\delta_{NP}(\delta_{NP}(x_0, v), \sigma)!$.

$\sigma \in \Sigma_o$. Then, $(\sigma, 0) \in \delta_{NP}(x_0, v).m$ because $\delta_{NP}(x_0, v).m = \delta_{NSP}(z_0, v).m$ and $(\sigma, 0) \in \delta_{NSP}(z_0, v).m$ (due to Definition 3.7-item 5)). So, due to Definition 3.10-item 5), $\delta_{NP}(\delta_{NP}(x_0, v), \sigma)!$.

Conclusion: By the principle of induction, $w \in L(NS||NP)$ is true for any $w \in L(NSP)$.

2) Take $w \in L(NS||NP)$, by induction, it is proved that $w \in L(NSP)$ is true.

Base case: Assume that $w = \epsilon \in L(NS||NP)$. Then, $w \in L(NSP)$ by definition.

Induction step: Let $w = v\sigma \in L(NS||NP)$ where the statement is true for v, i.e.,

$v \in L(NSP)$. It suffices to prove that the statement holds for $v\sigma$, i.e., $v\sigma \in L(NSP)$. Due to the definition of synchronous product (in [Cassandras and Lafortune 2009]), since $\Sigma_{NS} \subseteq \Sigma_{NSP}$, one can say any $w \in L(NS||NP)$ if $w \in L(NP)$ and $P_{\Sigma_{NS}}(w) \in L(NS)$. Due to Lemma B.4, $\delta_{NP}(x_0, v).a = \delta_G(a_0, P_{\Sigma_G}(v))$ and $\delta_{NP}(x_0, v\sigma).a = \delta_G(\delta_{NP}(x_0, v).a, \sigma)$.

Also, due to Lemma B.3, for $v \in L(NSP)$, $\delta_{NSP}(z_0, v).y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(v))$, and $\delta_{NSP}(z_0, v).a = \delta_G(a_0, P_{\Sigma_G}(v))$.

Moreover, since both $\delta_{NSP}(z_0, v)$ and $\delta_{NP}(x_0, v)$ are reachable through v, based on Lemma B.6, $\delta_{NSP}(z_0, v).m = \delta_{NP}(x_0, v).m$ and $\delta_{NSP}(z_0, v).l = \delta_{NP}(x_0, v).l$. Now, for different cases of $\sigma \in \Sigma_{NSP}$, we prove that $\delta_{NSP}(\delta_{NSP}(z_0, v), \sigma)!$.

$\sigma \in \Sigma_e$. Then, based on the assumption, $\delta_{NS}(y_0, P_{\Sigma_{NS}}(v)\sigma)!$, and so considering Definition 3.7-item 1), $\delta_{NSP}(\delta_{NSP}(z_0, v), \sigma)!$.

$\sigma \in \Sigma_c$. Then, $\delta_G(\delta_{NSP}(z_0, v).a, \sigma)!$ for the reason that $\delta_{NSP}(z_0, v).a = \delta_{NP}(x_0, v).a$ and $\delta_G(\delta_{NP}(x_0, v).a, \sigma)!$. Also, the condition $head(\delta_{NSP}(z_0, v).l) = (\sigma, 0)$ is satisfied since $\delta_{NSP}(z_0, v).l = \delta_{NP}(x_0, v).l$ and $head(\delta_{NP}(x_0, v).l) = (\sigma, 0)$ (based on Definition 3.10-item 2)). Hence, considering Definition 3.7-item 2), $\delta_{NSP}(\delta_{NSP}(z_0, v), \sigma)!$.

$\sigma \in \Sigma_{uc}$. Then, $\delta_G(\delta_{NSP}(z_0, v).a, \sigma)!$ for the reason that $\delta_{NSP}(z_0, v).a = \delta_{NP}(x_0, v).a$ and $\delta_G(\delta_{NP}(x_0, v).a, \sigma)!$, and so considering Definition 3.7-item 3), $\delta_{NSP}(\delta_{NSP}(z_0, v), \sigma)!$.

$\sigma = tick$. Then, $\delta_G(\delta_{NSP}(z_0, v).a, \sigma)!$ for the reason that $\delta_{NSP}(z_0, v).a = \delta_{NP}(x_0, v).a$ and $\delta_G(\delta_{NP}(x_0, v).a, \sigma)!$. Also, $\delta_{NS}(\delta_{NS}(y_0, P_{\Sigma_{NS}}(v)), \sigma)!$ due to the assumption. Moreover, $(\sigma, 0) \notin \delta_{NSP}(z_0, v).m$ for all $\sigma \in \Sigma_a$ since $(\sigma, 0) \notin \delta_{NP}(x_0, v).m$ (based on Definition 3.10-item 4)) and $\delta_{NSP}(z_0, v).m = \delta_{NP}(x_0, v).m$. Therefore, based on Definition 3.7-item 4), $\delta_{NSP}(\delta_{NSP}(z_0, v), \sigma)!$.

$\sigma \in \Sigma_o$. Then, $(\sigma, 0) \in \delta_{NSP}(z_0, v).m$ for the reason that $\delta_{NSP}(z_0, v).m = \delta_{NP}(x_0, v).m$ and $(\sigma, 0) \in \delta_{NP}(x_0, v).m$ (due to Definition 3.10-item 5). Also, $\delta_{NS}(\delta_{NS}(y_0, P_{\Sigma_{NS}}(v)), \sigma)!$ based on the assumption. So, due to Definition 3.7-item 5), $\delta_{NSP}(\delta_{NSP}(z_0, v), \sigma)!$.

Conclusion: By the principle of induction $w \in L(NSP)$ is true for any $w \in L(NS||NP)$. ■

Corollary B.1 (Lemma B.7). Given a plant G and a networked supervisor NS with event set Σ_{NS} such that $L(NS) \subseteq P_{\Sigma_{NS}}(L(NP))$:

1. $L(NSP) \subseteq L(NP)$, and
2. $L_m(NSP) \subseteq L_m(NP)$.

Proof. This clearly holds since due to Lemma B.7, $NSP = NS||NP$ and $\Sigma_{NS} \subseteq \Sigma_{NSP}$. ■

Lemma B.8 (Finite NP)**.** Given a plant G with a set of states A and a set of events Σ_G: NP is a finite automaton.

Proof. We need to prove that NP has a finite set of states and a finite set of events. Considering Definition 3.10, NP has a set of states $X = A \times Q' \times A' \times M \times L$. To prove that X is finite, it is sufficient to guarantee that A, Q', A', M and L are finite sets because as proved in [Jech 2013] the Cartesian product of finite sets is finite. A is finite as the

plant is assumed to be given as a finite automaton. A' is finite since for each $a' \in A'$, $a' \subseteq A$, and A is finite. $M(L)$ is finite as the maximum size of every element of M is limited to a finite number $M_{max}(L_{max})$. Moreover, $\Sigma_{NSP} = \Sigma_e \cup \Sigma_o \cup \Sigma_G$ is finite since G is a finite automaton, and so Σ_G is finite. Σ_e and Σ_o are finite since due to Definition 3.4, the size of Σ_e is equal to the size of Σ_c, and the size of Σ_o is equal to the size of Σ_a. ■

Lemma B.9 (Nonblocking NS)**.** The networked supervisor *NS* synthesized from Algorithm 3.1 is nonblocking.

Proof. Based on Property 3.3, Algorithm 3.1 terminates, let say after n iterations. Then, either $x_0 \in \mathit{Uncon}(\mathit{BS}(n))$ or $\mathit{BS}(n) = \varnothing$ where $\mathit{BS}(n) = \mathit{BPre}(\mathit{NS}(n) \cup \mathit{BLock}(\mathit{NS}(n)) \cup \mathit{TLock}(\mathit{NS}(n)))$. In the case that $x_0 \in \mathit{Uncon}(\mathit{BS}(n))$, the algorithm gives no result. Otherwise, the algorithm gives $\mathit{NS} = P_{\Sigma_{NS}}(\mathit{NS}(n))$ where $\mathit{NS}(n)$ is nonblocking since $\mathit{BLock}(\mathit{NS}(n)) = \varnothing$. Moreover, due to Lemma B.1, the projection preserves nonblockingness, and so *NS* is nonblocking. ■

Lemma B.10 (TLF NS)**.** The networked supervisor *NS* synthesized for a plant G using Algorithm 3.1 is TLF.

Proof. The proof is similar to the proof of Lemma B.9. ■

B.2 Proofs of Properties and Theorems

B.2.1 Proof of Property 3.1

It suffices to prove that $w \in L(G)$ for any $w \in P_{\Sigma_G}(L(\mathit{NSP}))$. Take arbitrary $w \in P_{\Sigma_G}(L(\mathit{NSP}))$. Then, according to Definition 3.1, $P_{\Sigma_G}(w') = w$ for some $w' \in L(\mathit{NSP})$. Then, due to Lemma B.3, $\delta_{\mathit{NSP}}(z_0, w').a = \delta_G(a_0, P_{\Sigma_G}(w'))$ meaning that $w \in L(G)$.

B.2.2 Proof of Property 3.2

The proof consists of two cases:

1) for any $w \in P_{\Sigma_G}(L(\mathit{NP}))$: $w \in L(G)$. This is proved by induction on the structure of w.

Base case: Assume $w = \epsilon$. Then, $w \in L(G)$ by definition.

Induction step: Assume that $w = v\sigma$ for some $v \in \Sigma_G^*$ and $\sigma \in \Sigma_G$ where the statement holds for v, i.e., $v \in L(G)$. It suffices to prove that the statement holds for $v\sigma$, i.e., $v\sigma \in L(G)$. Due to the projection properties, for $v\sigma \in P_{\Sigma_G}(L(\mathit{NP}))$, one can say there exists $v' \in \Sigma_{\mathit{NSP}}^*$, $P_{\Sigma_G}(v') = v\sigma$. Without loss of generality, let say $v' = v''\sigma$ where $P_{\Sigma_G}(v'') = v$. Then, due to Lemma B.4, $\delta_{\mathit{NP}}(x_0, v'').a = \delta_G(a_0, P_{\Sigma_G}(v'')) = \delta_G(a_0, v)$, and $\delta_{\mathit{NP}}(\delta_{\mathit{NP}}(x_0, v''), \sigma).a = \delta_G(a_0, P_{\Sigma_G}(v''\sigma)) = \delta_G(\delta_G(a_0, v), \sigma)$. So, $\delta_G(\delta_G(a_0, v), \sigma)!$ and the statement holds for $v\sigma$.

Conclusion: By the principle of induction, the statement $w \in L(G)$ holds for all $w \in P_{\Sigma_G}(L(\mathit{NP}))$.

2) If $max_c \leq L_{max}$, for any $w \in L(G)$: $w \in P_{\Sigma_G}(L(NP))$. This is proved by using induction on the structure of w.

Base case: assume $w = \epsilon$. Then $w \in P_{\Sigma_G}(L(NP))$ by definition.

Induction step: assume that $w = v\sigma$ for some $v \in L(G)$ and $\sigma \in \Sigma_G$ where the statement holds for v, i.e., $v \in P_{\Sigma_G}(L(NP))$. It suffices to prove that $v\sigma \in P_{\Sigma_G}(L(NP))$. For $v \in P_{\Sigma_G}(L(NP))$, there exists $v' \in \Sigma^*_{NSP}$, $P_{\Sigma_G}(v') = v$ due to the projection properties. Considering Definition 3.10, one of the following cases may occur at $\delta_{NP}(x_0, v')$.

$\sigma \in \Sigma_{uc}$, then due to item 3), $\delta_{NP}(\delta_{NP}(x_0, v'), \sigma)!$ because $\delta_G(\delta_{NP}(x_0, v').a, \sigma)!$. Applying the projection on $v'\sigma \in L(NP)$ results in $v\sigma \in P_{\Sigma_G}(L(NP))$.

$\sigma \in \Sigma_c$, then $(\sigma, 0) \in \delta_{NP}(x_0, v').l$ since due to Lemma B.5, N_c *ticks* earlier, σ_e was enabled in NP. When σ_e occurred, based on item 1), (σ, N_c) was certainly put in l as Assumption 2 holds. The occurrence of each *tick* (from N_c *ticks*) causes $l - 1$ as item 4) says. Also, the control channel is FIFO (l is a list), so even if a sequence of events have been enabled simultaneously, the ordering is preserved in l. So far, $head(\delta_{NP}(x_0, v').l) = (\sigma, 0)$ and $\delta_G(\delta_{NP}(x_0, v').a, \sigma)!$ as assumed. So, due to item 2), $v'\sigma \in L(NP)$, and $v\sigma \in P_{\Sigma_G}(L(NP))$.

$\sigma = tick$, then let us first empty $\delta_{NP}(x_0, v').m$ from any $(\sigma', 0)$ by executing $v_o \in \Sigma^*_o$. Then, $(\sigma', 0) \notin \delta_{NP}(x_0, v'\, v_o).m$. Also, $\delta_{NP}(x_0, v'\, v_o).a = \delta_{NP}(x_0, v').a$ since the execution of observed events only changes $\delta_{NP}(x_0, v').m$. $\delta_G(\delta_{NP}(x_0, v').a, tick)!$ due to the assumption, and so $\delta_G(\delta_{NP}(x_0, v'\, v_o).a, tick)!$. Now, as the worst case, assume that at $\delta_{NP}(x_0, v'\, v_o).a'$, only $v_c \in \Sigma^{*a}_c$ is enabled, and after that either *tick* occurs or nothing. Based on item 4) and item 1), this is then only v_{c_e} executed at $\delta_{NP}(x_0, v'\, v_o)$. $\delta_G(\delta_{NP}(x_0, v'\, v_o\, v_{c_e}).a, tick))!$, $(\sigma, 0) \notin \delta_{NP}(x_0, v'\, v_o\, v_{c_e}).m$, and $\neg\delta'_G(\delta_{NP}(x_0, v'\, v_o\, v_{c_e}).a', \sigma')!$ for all $\sigma, \sigma' \in \Sigma_a$. So, based on item 4), $v'\, v_o\, v_{c_e}\, tick \in L(NP)$, and so $v\, tick \in P_{\Sigma_G}(L(NP))$.

Conclusion: By the principle of induction, the statement $(w \in P_{\Sigma_G}(L(NP)))$ holds for all $w \in L(G)$.

B.2.3 Proof of Property 3.3

Algorithm 3.1 terminates if at some iteration i, $y_0 \in Uncon(bs(i))$ or $bs(i) = \varnothing$. At each iteration i, $bs(i) \subseteq Y$ since initially $bs(0) = BS(ns(0))$ where $BS(ns(0)) = BLock(ns(0)) \cup TLock(ns(0)$, and so $bs(0) \subseteq Y$ by definition. Also, $bs(i)$ is updated at Algorithm 3.1-line 15 to $BPre(ns(i)) \cup BS(ns(i))$ where $BPre(ns(i)) \subseteq Y$ and $BS(ns(i)) \subseteq Y$ by definition, and so $bs(i) \subseteq Y$. Since Y is a finite set, it suffices to prove that at each iteration, at least one state is removed from Y. Then, it is guaranteed that the algorithm loops finitely often. So, let's say $y_0 \notin Uncon(bs(i))$ and $bs(i) \neq \varnothing$ (because otherwise the algorithm terminates immediately). Then, there exists some state $y' \in bs(i)$. By definition this gives $y' \in Uncon(bs(i))$. Also, since at the end of each iteration, the automaton is made reachable (Algorithm 3.1-line 14), y' is reachable from y_0 (possibly through some intermediate states). According to Algorithm 3.1-line 12, at least y' is removed from Y, and so the algorithm terminates.

B.2.4 Proof of Theorem 3.1

We need to prove that for all $z \in Reach(z_0)$, there exists a $w \in \Sigma^*_{NSP}$ such that $\delta_{NSP}(z, w) \in Z_m$. Take $z \in Reach(z_0)$, then we need to find $w \in \Sigma^*_{NSP}$ for which $\delta_{NSP}(z, w) \in Z_m$. Let us assume that z is reachable from z_0 via $w_0 \in \Sigma^*_{NSP}$, i.e., $\delta_{NSP}(z_0, w_0) = z$. Then, due to Lemma B.3, $z.y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(w_0))$. Due to Lemma B.9, for $z.y \in Reach(y_0)$, there exists some $v \in \Sigma^*_{NS}$ such that $\delta_{NS}(z.y, v) \in Y_m$. Moreover, due to Algorithm 3.1-line 14, $L(ns(i)) \subseteq L(ns(i-1))$, and $ns(0) = NP$. Hence, $L(NS) \subseteq L(P_{\Sigma_{NS}}(NP))$ (Algorithm 3.1-line 20). Then, due to the projection properties, for $P_{\Sigma_{NS}}(w_0)v \in L(NS)$, one can say there exists some $w' \in L(NP)$, $P_{\Sigma_{NS}}(w') = P_{\Sigma_{NS}}(w_0)v$ such that $\delta_{NP}(x_0, w') \in X_m$ (due to the projection properties, any state y is marked only if $y \cap X_m \neq \varnothing$). Without loss of generality, assume that $w' = w'_0 w'_1$ for some $w'_0, w'_1 \in \Sigma^*_{NSP}$ with $P_{\Sigma_{NS}}(w'_0) = P_{\Sigma_{NS}}(w_0)$ and $P_{\Sigma_{NS}}(w'_1) = v$. Let $x'_1 \in X$ be such that $\delta_{NP}(x_0, w'_0) = x'_1$, and then $\delta_{NP}(x'_1, w'_1) \in X_m$. Moreover, due to Corollary B.1, $w_0 \in L(NP)$, and so $\delta_{NP}(x_0, w_0) = x_1$ for some $x_1 \in X$. So far, we have x_1, x'_1 are reachable from x_0 via w_0, w'_0, respectively, where $P_{\Sigma_{NS}}(w_0) = P_{\Sigma_{NS}}(w'_0)$. Thereto, x_1 is observationally equivalent to x'_1. Then, $x_1 \notin Uncon(BS(ns(i))$ at any iteration i because otherwise $x'_1 \in OBS(Uncon(BS(ns(i))))$, and w'_0 will be undefined ($y_0 \in Y \setminus Uncon(BS(ns(i))$, and so there exists at least a controllable event leading x_0 to x'_1 which is undefined). This is the case for all other states observationally equivalent to x_1 (because otherwise $P_{\Sigma_{NS}}(w_0) \notin L(NS)$ which contradicts the assumption). Therefore, $x_1 \notin Uncon(BS(ns(i))$ for any iteration i of the algorithm. So, at each iteration i, there exists a $w \in \Sigma^*_{NSP}$ leading x_1 to a marked state which does not become undefined because if it does, then $x_1 \in Uncon(BS(NS(i+1))$ which is a contradiction.

B.2.5 Proof of Theorem 3.2

We need to prove that for all $z \in Reach(z_0)$, there exists a $w \in \Sigma^*_{NSP}$ such that $\delta_{NSP}(z, w\,tick)!$. Take $z \in Reach(z_0)$, and assume z is reachable from z_0 via $w_0 \in \Sigma^*_{NSP}$, i.e., $\delta_{NSP}(z_0, w_0) = z$. Then, due to Lemma B.3, $z.a = \delta_G(a_0, P_{\Sigma_G}(w_0))$ and $z.y = \delta_{NS}(y_0, P_{\Sigma_{NS}}(w_0))$. Based on Definition 3.7, we need to find $w \in \Sigma^*_{NSP}$ such that $\delta_G(z.a, P_{\Sigma_G}(w)\,tick)!$, $\delta_{NS}(z.y, P_{\Sigma_{NS}}(w)\,tick)!$, and $(\sigma, 0) \notin m$ for all $\sigma \in \Sigma_a$. As guaranteed by Lemma B.10, NS is TLF, and so for $z.y \in Reach(y_0)$, there exists $v \in \Sigma^*_{NS}$ such that $\delta_{NS}(z.y, v\,tick)!$. Also, $L(NS) \subseteq P_{\Sigma_{NS}}(L(NP))$ (as stated before), and so from the projection properties, one can say there exists $v' \in \Sigma^*_{NSP}$, $P_{\Sigma_{NS}}(v') = v$, $\delta_{NP}(x, v'\,tick)!$. Let us take $w = v'$ for which we already know $\delta_{NS}(y, P_{\Sigma_{NS}}(w)\,tick)!$. Also, $(\sigma, 0) \notin m$ for all $\sigma \in \Sigma_a$ because otherwise Definition 3.10-item 4) could not be satisfied. It now suffices to prove $\delta_G(z.a, P_{\Sigma_G}(w)\,tick)!$. As Property 3.2 says, $P_{\Sigma_G}(L(NP)) \subseteq L(G)$, and so $P_{\Sigma_G}(w)\,tick \in L(G)$ for $w\,tick \in L(NP)$.

B.2.6 Proof of Theorem 3.3

We need to prove that if we take any $w \in L(NSP)$ and $u \in \Sigma_{uc} \cup \{tick\}$ such that $P_{\Sigma_G}(w)u \in L(G)$. Then, $wu \in L(NSP)$ for $u \in \Sigma_{uc}$, and for $u = tick$ when there does not exist any $\sigma_f \in \hat{\Sigma}_{for} \cup \Sigma_o$ such that $w\sigma_f \in L(NSP)$.

Take $w \in L(NSP)$ and $u \in \Sigma_{uc}$. From Lemma B.3, $\delta_{NSP}(z_0, w).a = \delta_G(a_0, P_{\Sigma_G}(w))$.

Based on Definition 3.7-item 2), u occurs only if it is enabled by G. As a result, $\delta_{NSP}(\delta_{NSP}(z_0, w), u)!$ since $\delta_G(\delta_{NSP}(z_0, w).a, u)!$ due to the assumption.

Take $u = tick$ where $\nexists_{\sigma \in \hat{\Sigma}_{for} \cup \Sigma_o}\ w\sigma \in L(NSP)$. Considering Definition 3.7-item 4), *tick* occurs in NSP after w if the following conditions hold; 1. $P_{\Sigma_G}(w)\, tick \in L(G)$, 2. $P_{\Sigma_{NS}}(w)\, tick \in L(NS)$, and 3. $\nexists \sigma \in \Sigma_o, \delta_{NSP}(z_0, w\sigma)!$. The first and the last conditions hold based on the assumption. So, we only need to prove $P_{\Sigma_{NS}}(w)\, tick \in L(NS)$. Due to Corollary B.1, for $w \in L(NSP)$: $w \in L(NP)$. Due to Property 3.2, for $P_{\Sigma_G}(w)\, tick \in L(G)$, there exists $w' \in L(NP)$, $P_{\Sigma_G}(w') = P_{\Sigma_G}(w)$ such that $w'.tick \in L(NP)$. Considering Definition 3.10, $w\, tick \in L(NP)$ for the following reasons; 1. $\delta_{NP}(x_0, w).a = \delta_{NP}(x_0, w').a$ and $\delta_{NP}(x_0, w).a' = \delta_{NP}(x_0, w').a'$ since $P_{\Sigma_G}(w') = P_{\Sigma_G}(w)$. Hence, $\delta_G(\delta_{NP}(x_0, w).a, tick)!$ and $\delta_G(\delta_{NP}(x_0, w).a', tick)!$ (since $\delta_{NP}(x_0, w'\, tick)!$). 2. $m \in M$ changes only by the execution of $\sigma \in \Sigma_G$. So, $\delta_{NP}(x_0, w).m = \delta_{NP}(x_0, w').m$ since $P_{\Sigma_G}(w') = P_{\Sigma_G}(w)$. Also, $(\sigma, 0) \notin \delta_{NP}(x_0, w').m$ for any $\sigma \in \Sigma_a$ since $\delta_{NP}(x_0, w'\, tick)!$, and so $(\sigma, 0) \notin \delta_{NP}(x_0, w).m$ for any $\sigma \in \Sigma_a$. Due to the assumption, $w\sigma \notin L(NSP)$ for $\sigma \in \Sigma_{for} \cup \Sigma_e \cup \Sigma_o$. Also, due to Theorem B.7, $NSP = NS||NP$. In the case that $w\sigma \in L(NP)$ for some $\sigma \in \Sigma_{for} \cup \Sigma_o$, then, due to Algorithm 3.1-line 5, it could not be disabled by NS. Also, if $w\sigma \in L(NP)$ for some $\sigma \in \Sigma_e$ where both *tick* and σ become disabled by NS, then by definition, $\delta_{NP}(x_0, w) \in BPre(NS)$ and will be removed which violates the assumption ($w \in L(NSP)$). Hence, $w\, tick \in L(NP)$ and *tick* does not become disabled by Algorithm 3.1, and so $P_{\Sigma_{NS}}(w)\, tick \in L(NS)$.

B.2.7 Proof of Theorem 3.4

To prove that NS is (timed networked) maximally permissive for G, we need to ensure that for any other proper networked supervisor (say NS') in the same NSC framework (with event set Σ_{NS}): $P_{\Sigma_G}(L({NS'}_{N_c} \|_{N_o} G)) \subseteq P_{\Sigma_G}(L(NSP))$. First, assume that $L(NS') \nsubseteq P_{\Sigma_{NS}}(L(NP))$. Then, any extra transition of NS' that is not included in $P_{\Sigma_{NS}}(L(NP))$ does not add any new transition to $P_{\Sigma_G}(L({NS'}_{N_c} \|_{N_o} G))$. Let say $v\sigma \in L(NS')$ and $v \in P_{\Sigma_{NS}}(L(NP))$, but $v\sigma \notin P_{\Sigma_{NS}}(L(NP))$ for $\sigma \in \Sigma_{NS}$. Also, there exists $w \in L({NS'}_{N_c} \|_{N_o} G)$ with $P_{\Sigma_{NS}}(w) = v$. If $\sigma = tick$, then σ cannot be executed in ${NS'}_{N_c} \|_{N_o} G$ because based on Definition 3.7-item 4), *tick* should be enabled by G which is not the case; *tick* is not enabled in NP, and so due to Property 3.2, it is not enabled in G. If $\sigma \in \Sigma_o$, then it does not matter if σ occurs in ${NS'}_{N_c} \|_{N_o} G$ because it does not change $P_{\Sigma_G}(L({NS'}_{N_c} \|_{N_o} G))$. If $\sigma \in \Sigma_e$, then as Lemma B.5 says, NP enables all enabling events of Σ_c that are executed in the plant on time (N_c *ticks* ahead). So, any extra enabling event by NS' will not be executed by the plant, and so it does not enlarge $P_{\Sigma_G}(L({NS'}_{N_c} \|_{N_o} G))$. Therefore, we continue the proof for the case that $L(NS') \subseteq P_{\Sigma_{NS}}(L(NP))$ (where Lemma B.7 and Corollary B.1 hold for NS'). Take an arbitrary $w \in P_{\Sigma_G}(L({NS'}_{N_c} \|_{N_o} G))$, it suffices to prove that $w \in P_{\Sigma_G}(L(NSP))$. Let say ${NS'}_{N_c} \|_{N_o} G = (z'_0, \Sigma_{NSP}, \delta_{NS'P}, Z'_m)$. For $w \in P_{\Sigma_G}(L({NS'}_{N_c} \|_{N_o} G))$, due to the projection properties, there exists $v' \in L({NS'}_{N_c} \|_{N_o} G)$ such that $P_{\Sigma_G}(v') = w$ where $\delta_{NS'P}(z'_0, v')$ is a TLF and non-blocking state (NS' is proper due to the assumption). Also, any uncontrollable active event/non-preemptable *tick* enabled at $\delta_G(a_0, w)$ is enabled at $\delta_{NS'P}(z_0, v')$, and it leads to a nonblobking and TLF state. Based on Lemma B.3, $P_{\Sigma_{NS}}(v') \in L(NS')$ for $v' \in L({NS'}_{N_c} \|_{N_o} G)$, and due to Corollary B.1, $v' \in L(NP)$. Moreover, due to Lemma B.7, $L({NS'}_{N_c} \|_{N_o} G) = L(NS'||NP)$, so regarding the definition of synchronous

product, for any $w' \in L(NP)$ and $P_{\Sigma_{NS}}(w') = P_{\Sigma_{NS}}(v')$: $w' \in L(NS'_{N_c} \|_{N_o} G)$. $\delta_{NS'P}(z'_0, w')$ is a TLF and non-blocking state because $NS'_{N_c} \|_{N_o} G$ is nonblocking and TLF due to the assumption. Also, any uncontrollable active event or non-preemptable *tick* enabled at w' leads to a nonblocking and TLF state since NS' is controllable for G by the assumption. Therefore, one can say $\delta_{NP}(x_0, v') \notin OBS(Uncon(BS(NP))$. Considering Algorithm 3.1, initially, $ns(0) = NP$ where $v' \in L(NP)$ and $\delta_{NS}(y_0, v') \notin OBS(Uncon(BS(ns(0)))$. The last statement holds for any iteration of the algorithm until the last one (say n) so that $\delta_{NS}(y_0, v') \notin OBS(Uncon(BS(NS(n)))$ because otherwise all $y \in OBS(Uncon(BS(NS(n)))$ are removed (based on Algorithm 3.1-line 6), and so $P_{\Sigma_{NS}}(v') \notin L(NS)$ because it leads $NS||NP$ ($NS||NP = NS(n)$) to a state in $Uncon(BS(NS(n)))$. Then, based on Lemma B.7, NSP becomes blocking/time-lock/uncontrollable which violates the assumption. Hence, (considering Algorithm 3.1-line 6) v' is not undefined by Algorithm 3.1, and so $P_{\Sigma_{NS}}(v') \in L(NS)$. Based on Lemma B.7, $L(NSP) = L(NS||NP)$. $P_{\Sigma_{NS}}(v') \in L(NS)$ and $v' \in L(NP)$, so $v' \in L(NSP)$ where applying the projection on Σ_G gives $w \in P_{\Sigma_G}(L(NSP))$.

B.2.8 Proof of Theorem 3.5

To simplify, let us denote $G||R^{\perp}$ by G^t, the networked plant $\Pi(G^t, N_c, N_o, L_{max}, M_{max})$ by NP^t and the networked supervised plant $NS_{N_c} \|_{N_o} G^t$ by NSP^t. We need to prove that if we take any $w \in P_{\Sigma_R}(L(NSP^t))$: $w \in L(R)$. Take $w \in P_{\Sigma_R}(L(NSP^t))$, then due to Definition 3.1, there exists $w' \in L(NSP^t)$ such that $P_{\Sigma_R}(w') = w$. Also, based on Property 3.1, $P_{\Sigma_G}(L(NSP^t)) \subseteq L(G^t)$, and so $P_{\Sigma_G}(w') \in L(G||R^{\perp})$. Applying the projection on Σ_R gives $P_{\Sigma_R}(w') \in L(R^{\perp})$. For $w \in P_{\Sigma_R}(L(NSP^t)) \cap L(R^{\perp})$, $w \in L(R)$ since the blocking state q_d added to $G||R$ to make $G||R^{\perp}$ is removed by NS as guaranteed by Theorem 3.1.

Appendix C

Proofs of Chapter 4

C.1 Technical Lemmas

Lemma C.1 (Algorithm termination)**.** Given a τ-automaton P, Algorithm 4.1 terminates.

Proof. The algorithm terminates when $y_0 \in BS$ or $BS = \varnothing$. Let say there exists at least a state $y' \in BS$ where $y' \neq y_0$ (otherwise the algorithm terminates immediately). Initially, $BS = BLock(S)$, and so $BS \subseteq Y$ by definition. At the end of each iteration BS is updated at Algorithm 4.1-line 12, where both $BLock(S) \subseteq Y$ and $Y_f \setminus Y_f(S) \subseteq Y$ by definition. So $BS \subseteq Y$, and it suffices to prove that at each iteration at least one state is removed from Y. This can be seen at Algorithm 4.1-line 10, where $Uncon_S(BS)$ is removed from Y, and $Uncon_S(BS)$ includes at least one state as BS is nonempty. ■

Lemma C.2 (Nonblocking and Controllable S)**.** Given a τ-automaton P with the set of events $\Sigma \cup \{\tau\}$, uncontrollable events Σ_{uc}, and forcible events Σ_{for}, Algorithm 4.1 results in a controllable S for P, and $S||P$ is nonblocking.

Sketch of the Proof. According to Lemma C.1, the algorithm terminates and it either gives no result (when $y_0 \in BS$) or a nonblocking S ($BS = \varnothing$ which means that $BLock(S) = \varnothing$ since $BLock(S) \subseteq BS$). S is controllable for P since the algorithm starts from $S = P$ and $BS = BLock(S)$. At Algorithm 4.1-line 10, all states belonging to $Uncon_S(BS)$ are removed. So, if there exists a state y reaching to a blocking state in an uncontrollable manner (a sequence of uncontrollable events or not preemptable τ), then $y \in Uncon_S(BS)$. In the case that a τ-transition enabled at some state y has been preempted by a forcible event, and later the forcible event becomes disabled, then y is added to BS at Algorithm 4.1-line 12. ■

C.2 Proofs of Properties and Theorems

C.2.1 Proof of Theorem 4.1

[Sketch of the Proof]

• S^t is controllable for G. Considering Definition 4.7, controllability actually means that the guard of an uncontrollable event and the invariant of a location where no forcible event is enabled do not change in $S^t||G$. The guard of an edge or the invariant of a location of G is modified in S^t only when a transition of P becomes undefined by S. Due to Lemma C.2, S is controllable for P meaning that no uncontrollable event or uncontrollable (not preemptable) τ is disabled by S. Hence, S^t is controllable for G.

• $S^t||G = S^t$ is nonblocking. First, $S^t||G = S^t$ for the following reason: suppose that $\delta(q_s, \sigma) = q_t$ is a controllable event transition of P that is disabled by P. Then, based on Definition 4.10, $q_t.R$ will be excluded from all possible regions allowed for taking the event σ in G, and so the guard of the edge labeled by σ will be restricted in S^t. In a similar way, whenever a τ-transition is disabled by S, the region of the target state will be excluded from the set of regions satisfied by the original invariant, and so for any location, S^t may restrict the invariant. In addition, based on Lemma C.2, S is nonblocking, and so any reachable state q in S is nonblocking. Considering Definition 4.9, q is nonblocking whenever $q.l$ is nonblocking. Therefore, any reachable state of S^t is nonblocking.

Appendix D

Proofs of Chapter 5

D.1 Technical Lemmas

Lemma D.1 (G-Clock Constraint)**.** φ is a G-clock constraint iff for any pair of clock valuations u_1, u_2, represented by the same clock region $r_G \in R_G$: $u_1 \models \varphi \iff u_2 \models \varphi$.

Proof. The proof is trivial. ■

Lemma D.2 (Clock Valuations)**.** For any pair of clock valuations $u_1, u_2 \in r_G$ for some $r_G \in R_G$, if $u_1 + \Delta_1 \in r_\Delta$ for some $r_\Delta \in R_G$ and $\Delta_1 \in \mathbb{R}_{\geq 0}$: there exists $\Delta_2 \in \mathbb{R}_{\geq 0}$ such that $u_2 + \Delta_2 \in r_\Delta$.

Proof. As illustrated by Figure 5.3, from two valuations from the same region in each case any move to another region (by passage of time) from one of these valuations is easily mimicked from the other valuation (possibly for a different amount of time passage). ■

Remark. In the coming lemmas and proofs, we frequently use "this term represents a G-clock constraint". The meaning of this is that although the term may not necessarily satisfy G-clock constraints as given by Definition 5.12, there is G-clock constraint that is logically equivalent with it (which means that for any valuation the term and its G-clock constraint representation have the same value).

Lemma D.3 (Negation of G-Clock Constraint)**.** For any G-clock constraint φ, the negation $\neg\varphi$ also represents a G-clock constraint.

Proof. This is proved by induction on the structure of G-clock constraints.

Base cases:

- for the atomic G-clock constraints $x < n$ and $x - y < n$, their negations are $x \geq n$ and $x - y \geq n$, respectively;
- for the atomic G-clock constraints $x = n$ and $x - y = n$, their negations are $x > n \vee x < n$ and $x - y > n \vee x - y < n$, respectively;

- for the atomic G-clock constraints $x > n$ and $x - y > n$, their negations are $x \leq n$ and $x - y \leq n$, respectively.

Induction step: Consider the G-clock constraint $\varphi = \varphi_1 \lozenge \varphi_2$ for some G-clock constraints φ_1 and φ_2, and $\lozenge \in \{\wedge, \vee\}$, where the statement holds for φ_1 and φ_2, i.e., $\neg\varphi_1$ and $\neg\varphi_2$ also represent G-clock constraints. If $\lozenge = \vee$, then $\neg(\varphi_1 \lozenge \varphi_2) = \neg\varphi_1 \wedge \neg\varphi_2$. Also, if $\lozenge = \wedge$, then $\neg(\varphi_1 \lozenge \varphi_2) = \neg\varphi_1 \vee \neg\varphi_2$. Both $\neg\varphi_1$ and $\neg\varphi_2$ represent G-clock constraints as assumed, and the combination of any two G-clock constraints by $\wedge$ and $\vee$ is also a G-clock constraint. So, $\neg(\varphi_1 \lozenge \varphi_2)$ represents a G-clock constraint.

Conclusion: By the principle of induction, the claim of Lemma D.3 holds for any G-clock constraint φ. ■

Lemma D.4 (Reset Update of G-Clock Constraint)**.** For any G-clock constraint φ and any reset r, $\varphi[r]$ also represents a G-clock constraint.

Proof. This is proved by induction on the structure of G-clock constraints.

Bases cases:

- for the atomic G-clock constraints $x < n$ and $x - y < n$, $\varphi[r]$ represents the G-clock constraint φ if $x, y \notin r$. If $x, y \in r$, $\varphi[r]$ represents the G-clock constraint *true* if $n \neq 0$ and *false* if $n = 0$. For $x - y < n$, $\varphi[r]$ represents the G-clock constraint $y > n$ if only $x \in r$, and $x < n$ if only $y \in r$.
- for the atomic G-clock constraints $x = n$ and $x - y = n$, $\varphi[r]$ represents the G-clock constraint φ if $x, y \notin r$. If $x, y \in r$, $\varphi[r]$ represents the G-clock constraint *true* if $n = 0$ and *false* if $n \neq 0$. For $x - y = n$, $\varphi[r]$ represents the G-clock constraint $y = n$ if only $x \in r$, and $x = n$ if only $y \in r$.
- for the atomic G-clock constraints $x > n$ and $x - y > n$, $\varphi[r]$ represents the G-clock constraint φ if $x, y \notin r$. If $x, y \in r$, $\varphi[r]$ represents the G-clock constraint *false*. For $x - y > n$, $\varphi[r]$ represents the G-clock constraint $y < n$ if only $x \in r$, and $x > n$ if only $y \in r$.

Induction step: Consider the G-clock constraint $\varphi = \varphi_1 \lozenge \varphi_2$ for some G-clock constraints φ_1 and φ_2, and $\lozenge \in \{\wedge, \vee\}$, where the statement holds for φ_1 and φ_2, i.e., $\varphi_1[r]$ and $\varphi_2[r]$ also represent G-clock constraints. $(\varphi_1 \lozenge \varphi_2)[r] = \varphi_1[r] \lozenge \varphi_2[r]$ because the reset update does not change anything else than replacing all clock variables of r by zero. So, in $(\varphi_1 \lozenge \varphi_2)[r]$, the clock variables from r in both φ_1 and φ_2 are replaced by zero which can equivalently be represented by $\varphi_1[r] \lozenge \varphi_2[r]$. Since the combination of any two G-clock constraints by $\wedge$ and $\vee$ is also a G-clock constraint, $(\varphi_1 \lozenge \varphi_2)[r]$ represents a G-clock constraint.

Conclusion: By the principle of induction, the claim of Lemma D.4 holds for any G-clock constraint φ. ■

Lemma D.5 (Δ-Time Invariance for *NBP*)**.** Given G-clock constraints φ_1 and φ_2, $\exists\Delta\ \varphi_1^{\uparrow\Delta} \wedge\ \forall\delta \leq \Delta\ \varphi_2^{\uparrow\delta}$ represents a G-clock constraint.

Proof. Let us indicate $\exists\Delta\ \varphi_1^{\uparrow\Delta} \wedge\ \forall\delta \leq \Delta\ \varphi_2^{\uparrow\delta}$ by Φ. Take a clock valuation u_1 represented

by a clock region of G, say $r_G \in R_G$, such that $u_1 \models \Phi$. According to Lemma D.1, it suffices to prove that for any region r_G and any two clock valuations u_1 and u_2 represented by r_G, $u_1 \models \Phi$ iff $u_2 \models \Phi$. Because of symmetry considerations it suffices to prove that $u_1 \models \Phi$ implies $u_2 \models \Phi$. Let us assume $u_1 \models \Phi$. Then there exists some Δ_1 such that $u_1 \models \varphi_1^{\uparrow\Delta_1}$ and $u_1 \models \forall\delta \leq \Delta_1\ \varphi_2^{\uparrow\delta}$. It is proved that there always exists a Δ_2 for which $u_2 \models \varphi_1^{\uparrow\Delta_2}$ and $u_2 \models \forall\delta \leq \Delta_2\ \varphi_2^{\uparrow\delta}$. Let us say $u_1^{\uparrow\Delta_1} \in r_\Delta$ for some $r_\Delta \in R_G$. Then, based on Lemma D.2, there exists a $\Delta_2 \in \mathbb{R}_{\geq 0}$ such that $u_2^{\uparrow\Delta_2} \in r_\Delta$. Since $u_1^{\uparrow\Delta_1} \models \varphi_1$ and $u_1^{\uparrow\Delta_1}, u_2^{\uparrow\Delta_2} \in r_\Delta$, by Lemma D.1: $u_2^{\uparrow\Delta_2} \models \varphi_1$. It suffices to prove that for all $\delta \leq \Delta_2$: $u_2 \models \varphi_2^{\uparrow\delta}$. Take $\delta_2 \leq \Delta_2$, and assume that $u_2^{\uparrow\delta_2} \in r_\delta$, where r_δ can be r_G, r_Δ, or any region in between. Based on Lemma D.2, there exists a δ_1 such that $u_1^{\uparrow\delta_1} \in r_\delta$. We prove that $\delta_1 \leq \Delta_1$ by contradiction. Assume $\delta_1 > \Delta_1$. Then, r_δ already passed r_Δ, and this contradicts the fact that r_δ is either r_G, r_Δ, or any other region in between. So, $\delta_1 \leq \Delta_1$, and $u_2^{\uparrow\delta_2} \models \varphi_2$ because $u_1^{\uparrow\delta_1} \models \varphi_2$, and $u_1^{\uparrow\delta_1}, u_2^{\uparrow\delta_2} \in r_\delta$. ■

Lemma D.6 (Nonblocking Predicate)**.** Given a plant G, $N^i(l)$ computed by Algorithm 5.1 in each iteration i and for each location $l \in L$ represents a G-clock constraint.

Proof. We do the proof by induction on the number of iterations i.

Base case: $i = 0$. Then, $N^0(l)$ is either $I_G(l)$ or *false*, and in each case, this is a G-clock constraint by definition.

Induction step: Assume that the statement holds for i, i.e., $N^i(l)$ is a G-clock constraint for all $l \in L$. It suffices to prove that the statement holds for $i+1$, i.e., $N^{i+1}(l)$ is a G-clock constraint for all $l \in L$. Consider Algorithm 5.1-line 6, $N^{i+1} = ① \vee ② \vee ③$. It suffices to prove that each of ①, ②, and ③ is a G-clock constraint because then, the disjunction of them is also a G-clock constraint. ① is a G-clock constraint as assumed. ② is a G-clock constraint because g and $I_G(l')$ are G-clock constraints by definition, and $N^i(l')[r]$ is a G-clock constraint since $N^i(l')$ is a G-clock constraint as assumed, and the reset update represents a G-clock constraint according to Lemma D.4. Then, the conjunction of g, $I_G(l')$, and $N^i(l')[r]$ gives a G-clock constraint by definition. Finally, the big disjuction in ② is over a finite number of G-clock constraints as the number of edges is finite. ③ is a G-clock constraint because $I_G(l)$ is a G-clock constraint by definition, and $N^i(l')$ is a G-clock constraint as assumed. So, based on Lemma D.5, ③ represents a G-clock constraint.

Conclusion: By the principle of induction, the claim of Lemma D.6 holds for any iteration i and location $l \in L$. ■

Lemma D.7 (Δ-Time Invariance for *BSP*)**.** Given G-clock constraints φ_1, φ_2, and φ_3, $\exists\Delta\ \varphi_1^{\uparrow\Delta} \wedge \forall\delta \leq \Delta \left(\varphi_2^{\uparrow\delta} \wedge \forall\delta' \leq \delta\ \varphi_3^{\uparrow\delta'}\right)$ also represents a G-clock constraint.

Proof. Let us indicate $\exists\Delta\ \varphi_1^{\uparrow\Delta} \wedge \left(\forall\delta \leq \Delta\ \varphi_2^{\uparrow\delta} \wedge \forall\delta' \leq \delta\ \varphi_3^{\uparrow\delta'}\right)$ by Φ. Take a clock valuation u_1 represented by a clock region of G, say $r_G \in R_G$, such that $u_1 \models \Phi$. According to Lemma D.1, it suffices to prove that for any region r_G and any two clock valuations $u_1, u_2 \in r_G$: $u_1 \models \Phi$ iff $u_2 \models \Phi$. Because of symmetry considerations it suffices to prove that $u_1 \models \Phi$ implies $u_2 \models \Phi$.

Consider an arbitrary region $r_G \in R_G$ and arbitrary clock valuations $u_1, u_2 \in r$. Let us assume $u_1 \models \Phi$. Then there exists some Δ_1 such that $u_1 \models \varphi_1^{\uparrow\Delta_1}$, and $u_1 \models \forall\delta \leq \Delta_1\ \varphi_2^{\uparrow\delta} \wedge \forall\delta' \leq \delta\ \varphi_3^{\uparrow\delta'}$. It is proved that there always exists a Δ_2 for which $u_2 \models \varphi_1^{\uparrow\Delta_2}$, and $u_2 \models \forall\delta \leq \Delta_2\ \varphi_2^{\uparrow\delta} \wedge \forall\delta' \leq \delta\ \varphi_3^{\uparrow\delta'}$. Let us say that $u_1^{\uparrow\Delta_1} \in r_\Delta$ for some $r_\Delta \in R_G$. Then, based on Lemma D.2, there exists a $\Delta_2 \in \mathbb{R}_{\geq 0}$ such that $u_2^{\uparrow\Delta_2} \in r_\Delta$. Since $u_1^{\uparrow\Delta_1} \models \varphi_1$ and $u_1^{\uparrow\Delta_1}, u_2^{\uparrow\Delta_2} \in r_\Delta$, by Lemma D.1, we have $u_2^{\uparrow\Delta_2} \models \varphi_1$.

What remains to prove is that for all $\delta \leq \Delta_2$: $u_2 \models \varphi_2^{\uparrow\delta}$ and for all $\delta' \leq \delta$: $u_2 \models \varphi_3^{\uparrow\delta'}$. Take $\delta_2 \leq \Delta_2$, and assume $u_2^{\uparrow\delta_2} \in r_\delta$, where r_δ can be r_G, r_Δ, or any region in between. Then, for any $\delta' \leq \delta_2$, $u_2^{\uparrow\delta'}$ moves to either r_G, r_δ, or any region in between. Let us take $\delta_2' \leq \delta_2$, and assume that $u_2^{\uparrow\delta_2'}$ moves to $r_{\delta'}$. Based on Lemma D.2, there exists a δ_1 and a δ_1' such that $u_1^{\uparrow\delta_1} \in r_\delta$ and $u_1^{\uparrow\delta_1'} \in r_{\delta_1'}$. We prove that $\delta_1 \leq \Delta_1$ and $\delta_1' \leq \delta_1$ by contradiction. 1) Assume $\delta_1 > \Delta_1$. Then, r_δ already passed r_Δ, and this contradicts the fact that r_δ is either r_G, r_Δ, or any other region in between. 2) Assume $\delta_1' > \delta_1$. Then, $r_{\delta'}$ already passed r_δ, and this contradicts the fact that $r_{\delta'}$ is either r_G, r_δ, or any other region in between. ■

Lemma D.8 (Bad State Predicate)**.** Given a plant G and $NBP(G)$, $B^i(l)$ computed by Algorithm 5.2 in each iteration i and for each location $l \in L$ represents a G-clock constraint.

Proof. This is proved in a similar way to the proof of Lemma D.6, where Lemma D.7 is used to show that ⑥ represents a G-clock constraint. ■

Lemma D.9 (Adapted Guards)**.** Given a plant G, $e.g^m$ computed by Algorithm 5.3 in each iteration m and for each edge $e \in E_S$ represents a G-clock constraint.

Proof. We do the proof by induction on the number of iterations m.

Base case: $m = 0$. Then, $e.g^m = e.g$ for any $e \in E_S$ which is a G-clock constraint by definition.

Induction step: Assume that the statement holds for m, i.e., $e.g^m$ is a G-clock constraint for all $e \in E_S$. It suffices to prove that the statement holds for $m+1$, i.e., $e.g^{m+1}$ represents a G-clock constraint for all $e \in E_S$. Consider Algorithm 5.3-line 13, $e.g^{m+1} = e.g^m \wedge \neg B^{n,m}(l')[r]$. Now, $e.g^m$ is a G-clock constraint as assumed. $B^{n,m}(l')$ is a G-clock constraint because according to the proof of Lemma D.8, in each iteration, the bad state predicate of each location is a G-clock constraint. Based on Lemma D.4, $B^{n,m}(l')[r]$ represents a G-clock constraint, and so due to Lemma D.3, $\neg B^{n,m}(l')[r]$ represents a G-clock constraint. Then, the conjunction of $e.g^m$, and $\neg B^{n,m}(l')[r]$ gives a G-clock constraint by definition.

Conclusion: By the principle of induction, the claim of Lemma D.9 holds for any iteration m and edge $e \in E_S$. ■

Lemma D.10 (Adapted Invariants)**.** Given a plant G, $I_S^n(l)$ computed by Algorithm 5.3 in each iteration n and for each location $l \in L$ represents a G-clock constraint.

Proof. This is proved in a similar way to the proof of Lemma D.9. ■

Remark. As Algorithm 5.3 terminates (See Appendix D.2.5), in the coming proofs, for a given a plant G and $TSCS(G)$, it is assumed that the outer loop (Loop-2) terminates in $n = N$ iterations, and for each $0 \leq n \leq N$, the inner loop (Loop-1 inside Loop-2) terminates in $m = M_n$ iterations. S^{N,M_N} denotes the result of the final iteration, so that $S = S^{N,M_N}$ is the output of $TSCS(G)$.

Lemma D.11 (Synthesis Intermediate Results)**.** Given a plant G, the result of $TSCS(G)$ at any iteration n, m $(n \leq N, m \leq M_n)$ is a TA.

Proof. Initially, S is set to G, which is a TA. Then, at each iteration over n, m, only some of the guards and invariants may change. According to Lemma D.9 and Lemma D.10, the adapted guards and invariants are always clock constraints. So, based on Definition 5.3, the result of $TSCS(G)$ at any iteration n, m is a TA. ■

Remark. As Lemma D.11 holds, in the coming proofs, the result of Algorithm 5.3 at iteration n, m $(n \leq N, m \leq M_n)$ is assumed to be the TA $S^{n,m}$, represented by the automaton $(C, L, \Sigma_G, E_S^m, L_m, l_0, I_S^n)$, where E_S^m is the set of edges, and I_S^n gives the invariants of $S^{n,m}$.

D.2 Proofs of Properties and Theorems

D.2.1 Proof of Property 5.1

Based on Lemma D.6, in each iteration of the algorithm, say i, and for any location $l \in L$: $N^i(l)$ represents a G-clock constraint. At Algorithm 5.1-line 6, $N^i(l)$ is adapted to the G-clock constraint $N^{i+1}(l) = N^i(l) \vee (② \vee ③)$ for all $l \in L$. Both ② and ③ represent G-clock constraint as proved in Lemma D.6. So, $Z(N^{i+1}(l)) = Z(N^i(l)) \cup Z(② \vee ③)$ where $Z(② \vee ③) \in \mathcal{P}(R_G)$. Then, if $Z(② \vee ③) \subseteq Z(N^i(l))$: $Z(N^{i+1}(l)) = Z(N^i(l))$. So, $N^{i+1}(l) = N^i(l)$, and the algorithm terminates (Algorithm 5.1-line 8). Otherwise, at least a region $r_G \in R_G$ is added to $Z(N^i(l))$ so that $Z(N^{i+1}(l)) = Z(N^i(l)) \cup \{r_G\}$. Since L and R_G are both finite, this can occur only finitely many times.

D.2.2 Proof of Property 5.2

This property is proved in two parts:

1) take an arbitrary nonblocking state (l, u), and assume that from (l, u), a marked state can be reached in j transitions. Since the algorithm terminates and in each iteration (Algorithm 5.1-line 6), the nonblocking condition for a given location l is never strengthened, it always holds that $N^i(l) \Rightarrow N(l)$. So, to conclude that $u \models N(l)$, we prove that $u \models N^i(l)$ for some i by induction on j:

Base case: assume that from (l, u), a marked state is reached in 0 transitions. In other words, $l \in L_m$, and so $N^0(l) = I_G(l)$ by definition. Then, for $i = 0$, $u \models N^i(l)$ since

the semantic graph only contains states (l, u) for which the clock valuation satisfies the invariant of the location; $u \models I_G(l)$.

Induction step: assume that from (l, u), a marked state is reached in $j+1$ transitions. Also, assume that (l, u) leads to a state, say (l', u'), in one transition (this means that from (l', u'), a marked state is reached in j transitions), where the statement holds for (l', u') i.e., $u' \models N^i(l')$ for some i (by induction assumption). We prove that $u \models N^{i+1}(l)$.

If (l, u) moves to (l', u') by an event transition, say $\sigma \in \Sigma$, where this transition is related to an edge, (l, σ, g, r, l'). Then, based on Definition 5.7, $u \models g$, and $u[r] \models I_G(l')$. Also, $u[r] \models N^i(l')$ since $u' \models N^i(l')$ as assumed and $u' = u[r]$. So, $u \models N^{i+1}(l)$ since $u \models ②$. If (l, u) moves to (l', u') by a time transition, say Δ. Then, $u + \Delta \models N^i(l)$ because based on Definition 5.7, $l' = l$, $u' = u + \Delta$, and $u' \models N^i(l')$ as assumed. Also, for all $\delta \leq \Delta$: $u + \delta \models I_G(l)$ by definition. So, $u \models N^{i+1}(l)$ since $u \models ③$.

Conclusion: for any state (l, u) in the semantic graph of G that is nonblocking: $u \models N(l)$.

2) take an arbitrary (l, u) for which $u \models N(l)$. Since the algorithm terminates, and in each iteration the nonblocking condition for a given location l is never strengthened, there is always some i such that $u \models N^i(l)$. We prove by induction on i that from (l, u), a marked state is reached:

Base case: assume $u \models N^0(l)$. Then, $N^0(l)$ cannot be *false*, and so from (l, u), a marked state is reached (in 0 transitions) as $l \in L_m$.

Induction step: assume $u \models N^{i+1}(l)$, and the statement holds for i , i.e., for any (l', u') with $u' \models N^i(l')$: from (l', u'), a marked state is reached (induction assumption).

Considering the nonblocking predicate computation (line6), $u \models N^{i+1}(l)$ either because already $u \models N^i(l)$, or because $u \models ②$ or $u \models ③$.

If $u \models N^i(l)$. Then, from (l, u), a marked state is reached based on the induction assumption.

If $u \models ②$, then there exists at least one edge (l, σ, g, l', r) such that $u \models g \wedge I_G(l')[r] \wedge N^i(l')[r]$. Since $u \models N^i(l')[r]$ and $u' = u[r]$, $u' \models N^i(l')$. So, based on the induction assumption, from (l', u'), a marked state is reached. Also, since $u \models g \wedge I_G(l')[r]$, due to Definition 5.7, there is an event transition leading from (l, u) to (l', u'). So, from (l, u), a marked state is reached.

If $u \models ③$, then there exists Δ such that $u+\Delta \models N^i(l)$, and for all $\delta \leq \Delta$: $u+\delta \models I_G(l)[r]$. Since $u + \Delta \models N^i(l)$ with $u' = u + \Delta$, based on induction assumption, from (l', u'), a marked state is reached. Also, due to Definition 5.7, there is a time transition from (l, u) to (l', u') as for all $\delta \leq \Delta$: $u + \delta \models I_G(l)[r]$. So, from (l, u), a marked state is reached.

Conclusion: from any state (l, u) in the semantic graph of G with $u \models N(l)$, a marked state is reached.

D.2.3 Proof of Property 5.3

This property is proved in a similar way to the proof of Property 5.1, where Lemma D.7 is used to show that ⑥ represents a G-clock constraint.

D.2.4 Proof of Property 5.4

This property is proved in two parts:

1) take an arbitrary bad state (l, u), and assume that from (l, u) a blocking state can be reached in j (uncontrollable) transitions. Since the algorithm terminates, and in each iteration (Algorithm 5.2-line 6), the bad state condition for a given location l is never strengthened, it always holds that $B^i(l) \Rightarrow B(l)$. So, to conclude that $u \models B(l)$, we prove that $u \models B^i(l)$ for some i by induction on j:

Base case: from (l, u), a blocking state is reached in 0 transitions. Then, due to Property 5.2 $u \not\models N(l)$, and so for $i = 0$, $u \models B^i(l)$ by definition.

Induction step: from (l, u), a blocking state can be reached in $j + 1$ (uncontrollable) transitions. Assume that in one (uncontrollable) transition, (l, u) moves to a state, say (l', u'), where the statement holds for (l', u') i.e., $u' \models B^i(l')$ for some i (by the induction assumption). We prove that $u \models B^{i+1}(l)$.

If (l, u) moves to (l', u') by an uncontrollable event transition that is related to an edge (l, σ, g, r, l'). Then, based on Definition 5.7, $u \models g$, and $u[r] \models I_G(l')$. Also, $u[r] \models B^i(l')$ since $u' \models B^i(l')$ as assumed and $u' = u[r]$. So, $u \models B^{i+1}(l)$ since $u \models$ ⑤.

If (l, u) moves to (l', u') by a time transition, say Δ, that is not preemptable. Then, $u \models B^{i+1}(l)$ since $u \models$ ⑥ for the following reasons: 1) $u + \Delta \models B^i(l)$ because based on Definition 5.7, $l' = l$, $u' = u + \Delta$, and $u' \models B^i(l')$ as assumed, 2) for all $\delta \leq \Delta$: $u + \delta \models I_G(l)$ by definition, and 3) since Δ is not a preemptable time transition, there is no forcible event enabled at (l, u) so that the condition on forcible events always holds (is *true*).

Conclusion: for any bad state (l, u) in (the semantic graph of) G: $u \models B(l)$.

Assume that $u \models B^i(l)$. We prove by induction on i that from (l, u), a blocking state is reached within i uncontrollable transitions:

Base case: $u \models B^0(l)$. Then, $u \not\models N(l)$ by definition. So, based on Property 5.2, (l, u) is not a marked state, and any transition enabled at (l, u) does not lead to a nonblocking state. So, from (l, u), a blocking state is reached in 0 uncontrollable transitions.

Induction step: assume $u \models B^{i+1}(l)$, where the statement holds for i, i.e., for any (l', u') with $u' \models B^i(l')$: from (l', u'), a blocking state is by the induction assumption reached within i uncontrollable transitions.

Considering the bad state predicate computation, $u \models B^{i+1}(l)$ because already $u \models B^i(l)$, or because $u \models$ ⑤ or $u \models$ ⑥.

If $u \models B^i(l)$, then, (l, u) is a bad state based on the induction assumption.

If $u \models$ ⑤, then there exists at least one edge (l, σ, g, l', r), labeled by an uncontrollable

event, such that $u \models g \wedge I_G(l')[r] \wedge B^i(l')$. Since $u \models B^i(l')[r]$ and $u'[r] = u$, $u' \models B^i(l')$. So, based on the induction assumption, (l', u') is a bad state.

Also, since $u \models g \wedge I_G(l')[r]$, due to Definition 5.7, there is an uncontrollable event transition from (l, u) to (l', u'). So, (l, u) is a bad state.

If $u \models ⑥$, then there exists Δ such that $u+\Delta \models B^i(l)$, and for all $\delta \leq \Delta$: $u+\delta \models I_G(l)[r]$ (note that there is no forcible event that can preempt time). Since $u + \Delta \models B^i(l)$ with $u' = u + \Delta$, based on the induction assumption, (l', u') is a bad state.

Also, since $u+\delta \models I_G(l)[r]$ for all $\delta \leq \Delta$, due to Definition 5.7, there is a time transition from (l, u) to (l', u') that is not preemptable. So, (l, u) is a bad state.

Conclusion: for any state (l, u) in (the semantic graph of) G such that $u \models B(l)$: (l, u) is a bad state.

D.2.5 Proof of Property 5.5

Inside each iteration over n (loop-2), the iteration over m (loop-1) terminates because the computation of both $N^{n,m}$ and $B^{n,m}$ terminate due to property 5.1 and property 5.3, respectively. Also, whenever all the guards stay the same (Algorithm 5.3-line 16). Due to Lemma D.9, in each iteration m and for any edge $e \in E_S$, $e.g^m$ represents a clock constraint which is adapted to the clock constraint $e.g^{m+1} = e.g^m \wedge \neg B^{n,m}(l')[r]$ at Algorithm 5.3-line 13. Based on the properties stated for Z, $Z(e.g^{m+1}) = Z(e.g^m) \cap Z(\neg B^{n,m}(l')[r])$, and so $Z(e.g^{m+1}) \subseteq Z(e.g^m)$ for all $e \in E_S$. In the case that $Z(e.g^{m+1}) = Z(e.g^m)$ for all $e \in E_S$, the iteration over m terminates because $e.g^{m+1} = e.g^m$ for any $e \in E_S$. Otherwise, in each iteration, at least one region $r_G \in R_G$ is excluded from $Z(e.g^m)$ for some $e \in E_S$, i.e., $Z(e.g^{m+1}) = Z(e.g^m) \setminus \{r\}$ such that $r \notin Z(e.g^m)$, and so loop-1 can iterate only finitely often as E_S and R_G are both finite. The iteration over n (loop-2) terminates whenever all location invariants stay the same (Algorithm 5.3-line 25). Based on Lemma D.10, in each iteration of the algorithm n, and for any location $l \in L$: $I_S^n(l)$ represents a clock constraint which is adapted to $I_S^{n+1} = I_S^n(l) \wedge \neg B^{n,m}(l)$ at Algorithm 5.3-line 20. Then, for the same reason stated for termination of loop-1, loop-2 also terminates.

D.2.6 Proof of Property 5.6

The proof follows immediately from Lemma D.11.

D.2.7 Proof of Property 5.7

According to Lemma D.11, for any n, m ($n \leq N, m \leq M_n$): $S^{n,m}$ is a TA. Also, according to Property 5.6, S is a TA. To conclude that $S \subseteq G$, we prove that for all $n \leq N$, for all $m \leq M_n$: $S^{n,m} \subseteq G$ using nested induction on n and m. Then, in particular $S^{N,M_N} = TSCS(G) \subseteq G$, which is to be proven. Induction on n:

Base case: $n = 0$, and we prove that for all $m \leq M_0$: $S^{0,m} \subseteq G$ by induction on m:

- **Base case:** $S^{0,0} = G$, and so $S^{0,0} \subseteq G$.

- **Induction step:** assume $S^{0,m} \subseteq G$. Then, $S^{0,m+1}$ differs from $S^{0,m}$ only in terms of guards. So, considering Definition 5.5 and the construction of S in Algorithm 5.3, it only suffices to prove that for all $(l_s, \sigma, g_S^{m+1}, r, l_t) \in E_G^{m+1}$: $(l_s, \sigma, g_G, r, l_t) \in E_G$ for some g_G such that $g_S^{m+1} \Rightarrow g_G$.

  Take arbitrary edge $(l_s, \sigma, g_S^{m+1}, r, l_t) \in E_G^{m+1}$. Then, considering Algorithm 5.3-line 13, $(l_s, \sigma, g_S^m, r, l_t) \in E_S^m$ such that either $g_S^m = g_S^{m+1}$, or it is strengthened, and so $g_S^{m+1} \Rightarrow g_S^m$. Also, since $S^{0,m} \subseteq G$, then for $(l_s, \sigma, g_S^m, r, l_t) \in E_S^m$: $(l_s, \sigma, g_G, r, l_t) \in E_G$ for some g_G such that $g_S^m \Rightarrow g_G$. Thereto, for all $(l_s, \sigma, g_S^{m+1}, r, l_t) \in E_G^{m+1}$: $(l_s, \sigma, g_G, r, l_t) \in E_G$ for some g_G such that $g_S^{m+1} \Rightarrow g_G$.

- **Conclusion:** for all $m \leq M_0$: $S^{0,m} \subseteq G$.

Induction step: assume $S^{n,m} \subseteq G$ for all $m \leq M_n$. We prove that $S^{n+1,m} \subseteq G$ for all $m \leq M_{n+1}$ using induction on m:

- **Base case:** $S^{n,0} \subseteq G$ by assumption. $S^{n+1,0}$ differs from $S^{n,0}$ only in terms of invariants. So, considering Definition 5.5, it suffices to prove that for all $l \in L$: $I_S^{n+1}(l) \Rightarrow I_G(l)$. Take arbitrary $l \in L$. Then, $I_S^n(l) \Rightarrow I_G(l)$ since $S^{n,0} \subseteq G$. Considering Algorithm 5.3-line 20, at iteration $n+1$, either the invariant stays the same, or it is strengthened such that $I_S^{n+1}(l) \Rightarrow I_S^n(l)$. So, $I_S^{n+1}(l) \Rightarrow I_G(l)$ as $I_S^n(l) \Rightarrow I_G(l)$.

- **Induction step:** assume $S^{n+1,m} \subseteq G$ for all $m \leq M_{n+1}$. Then, $S^{n+1,m+1}$ differs from $S^{n+1,m}$ only in terms of guards. Then, $S^{n+1,m+1} \subseteq G$ for the same reason stated in the previous induction step on m.

- **Conclusion:** for all $m \leq M_n$: $S^{n,m} \subseteq G$.

Conclusion: for all $n \leq N$, for all $m \leq M_n$: $S^{n,m} \subseteq G$.

D.2.8 Proof of Property 5.8

Take arbitrary state (l, u) that is reachable in (the semantic graph of) S. We prove that $u \not\models B(l)$ by using induction on the length of the path from (l_0, u_0) to (l, u).

Base case: $(l, u) = (l_0, u_0)$. Then, we already have assumed that $u_0 \not\models B(l_0)$.

Induction step: Assume that (l, u) is reached from a (reachable) state, say (l', u') (by an event or time transition), where the statement holds for (l', u') (by induction assumption), i.e., $u' \not\models B(l')$. We prove that the statement holds for (l, u), i.e., $u \not\models B(l)$ for different cases of transitions from (l', u') to (l, u).

(l, u) is reached from (l', u') by $\sigma \in \Sigma_c$, and assume that this transition is related to an edge $(l', \sigma, g^{M_N}, r, l) \in E_S^{N,M_N}$. According to Definition 5.7, $u' \models g^{M_N}$. Also, based on Algorithm 5.3-line 13, g^{M_N} has been adapted in the last iteration such that $u' \models \neg B(l)[r]$, which is equivalent to $u'[r] \models \neg B(l)$. Again according to Definition 5.7, $u = u'[r]$, and so $u \models \neg B(l)$, which is equivalent to $u \not\models B(l)$.

(l, u) is reached from (l', u') by a time transition, say Δ, where $F_S(l) \neq \varnothing$. According to Definition 5.7, $u' + \Delta \models I^N(l)$. Based on Algorithm 5.3-line 20, $I_S^N(l)$ has been adapted in the last iteration such that $u' + \Delta \not\models B(l')$. Again according to Definition 5.7, $l' = l$,

and $u = u' + \Delta$. So, $u \not\models B(l)$

(l, u) is reached from (l', u') by $\sigma \in \Sigma_{uc}$, and assume that this transition is related to an edge $(l', \sigma, g^{M_N}, r, l) \in E_S^{N,M_N}$. Then, $u' \models g^{M_N} \wedge I_S^N(l)[r]$ by Definition 5.7. By contradiction, assume that $u \models B(l)$. Then, $u' \models B(l)[r]$ because $u' = u[r]$. So, $u' \models B(l')$ as $u' \models ⑤$ in the bad state predicate computation of l'. $u' \models B(l')$ contradicts the induction assumption, and consequently it must be the case that $u \not\models B(l)$ as required.

(l, u) is reached from (l', u') by a time transition, say Δ, where $F_S(l) = \varnothing$. Then, Also, $u' + \delta \models I_S^N(l)$ for all $\delta \leq \Delta$ by Definition 5.7. By contradiction, assume that $u \models B(l)$. Then $u' + \Delta \models B(l)[r]$ because $u' = u + \Delta$. Since $F_S(l) = \varnothing$, the condition on δ' in the bad state predicate computation of l' always gives true. As a result, $u' \models B(l')$ as $u' \models ⑥$ in the bad state predicate computation. $u' \models B(l')$ contradicts the induction assumption, and consequently it must be the case that $u \not\models B(l)$ as required.

Conclusion: for any reachable state (l, u) (in the semantic graph) of S: $u \not\models B(l)$.

D.2.9 Proof of Theorem 5.1

We need to prove that for any $w \in L(S||G)$ and $\sigma \in \Sigma_{uc} \cup \mathbb{R}_{\geq 0}$, whenever $w\sigma \in L(G)$, then $w\sigma \in L(S||G)$, or $\sigma \in \mathbb{R}_{\geq 0}$ and $w\sigma' \in L(S||G)$ for some $\sigma' \in \Sigma_{for}$. Consider arbitrary $w \in L(S||G)$ and $\sigma \in \Sigma_{uc} \cup \mathbb{R}_{\geq 0}$, and assume that $w\sigma \in L(G)$. Now assume that $\sigma \notin \mathbb{R}_{\geq 0}$ or $w\sigma' \notin L(S||G)$ for all $\sigma' \in \Sigma_{for}$. It suffices to prove that $w\sigma \in L(S||G)$. Since $S \subseteq G$ (based on Property 5.7), it suffices to prove $w\sigma \in L(S)$.

To conclude that $w\sigma \in L(S)$, we prove that for all $n \leq N$, for all $m \leq M_n$: $w\sigma \in L(S^{n,m})$ using nested induction on n and m. Induction on n:

Base case: $n = 0$. We prove that for all $m \leq M_0$, $w\sigma \in L(S^{0,m})$ using induction on m:

- **Base case:** $w\sigma \in L(S^{0,0})$ since $S^{0,0} = G$ and $w\sigma \in L(G)$ as assumed.
- **Induction step:** assume $w\sigma \in L(S^{0,m})$.

  $S^{0,m+1}$ may differ from $S^{0,m}$ only because the guards of some edges labeled by controllable events have been modified. Thereto, nothing changes in terms of the occurrence of an uncontrollable event or a time transition so that $w\sigma \in L(S^{0,m+1})$.
- **Conclusion:** for all $m \leq M_0$: $w\sigma \in L(S^{0,m})$.

Induction step: assume that for all $m \leq M_n$, $w\sigma \in L(S^{n,m})$. We prove that for all $m \leq M_{n+1}$, $w\sigma \in L(S^{n+1,m})$ using induction on m:

- **Base case:** we prove that $w\sigma \in L(S^{n+1,0})$.

  Since $w \in L(S||G)$, $w \in L(S)$, and so $w \in L(S^{n,m})$ for any n, m as S is the final result of the algorithm. For $w \in L(S^{n,0})$ (it holds by the induction assumption), assume that there exists some $l_s \in L$ and a clock valuation u_s such that (l_s, u_s) is reached from $(l_0, \mathbf{0})$ by w in (the semantic graph of) $S^{n,0}$. To conclude $w\sigma \in L(S^{n+1,0})$, we prove that σ occurs at (l_s, u_s) in $S^{n+1,0}$ for different cases of σ:

  $\sigma \in \Sigma_{uc}$. Based on the assumption, σ occurs at (l_s, u_s) in $S^{n,0}$. Assume that

σ transition is related to an edge $e = (l_s, \sigma, g, r, l_t)$ in $S^{n,0}$. Then, according to Definition 5.7: $u_s \models e.g^0$ and $u_s[r] \models I_S^n(l_t)$. So, $u_s \models e.g^0$, and it suffices to prove that $u_s[r] \models I_S^{n+1}(l_t)$. We continue the proof by contradiction. Assume that $u_s[r] \not\models I_S^{n+1}(l_t)$. Then, based on Algorithm 5.3-line 20, $u_s[r] \models B^{n,0}(l_t)$ because already $u_s[r] \models I_S^n(l_t)$ as assumed. Considering the computation of the bad state predicate of l_s, $u_s \models B^{n,0}(l_s)$ because $u_s \models e.g^0 \wedge I_S^n(l_t)[r] \wedge B^{n,0}(l_t)[r]$.

Since $u_s \models B^{n,0}(l_s)$, based on Property 5.4, (l_s, u_s) is a bad state, and this contradicts the assumption that (l_s, u_s) is reachable in S because then due to Property 5.8, (l_s, u_s) is not a bad state.

$\sigma = \Delta$, and there is no $\sigma' \in \Sigma_{for}$ such that $w\sigma' \in L(S||G)$. Then, for $w\sigma \in L(S^{n,0})$, according to Definition 5.7: $u_s + \delta \models I_S^n(l_s)$ for all $\delta \leq \Delta$. Also, the algorithm does not change the invariant as $F_S(l_s) = \varnothing$ so that $I_S^{n+1}(l_s) = I_S^n(l_s)$. So, σ occurs at (l_s, u_s) in $S^{n+1,0}$.

- **Induction step:** assume $w\sigma \in L(S^{n+1,m})$. Then, $w\sigma \in L(S^{n+1,m+1})$ for the same reason stated in the previous induction step on m.
- **Conclusion:** for all $m \leq M_n$, $w\sigma \in L(S^{n,m})$.

Conclusion: for all $n \leq N$, for all $m \leq M_n$: $w\sigma \in L(S^{n,m})$.

D.2.10 Proof of Theorem 5.2

First of all $L(S) \subseteq L(G)$, and so $L(S||G) = L(S)$. So, it suffices to prove that S is nonblocking, i.e., any reachable state in (the semantic graph of) S is nonblocking.

Take arbitrary state (l, u) that is reachable in (the semantic graph of) S. According to Property 5.8, $u \not\models B^{N,M_N}(l)$. This, based on Property 5.4, means that (l, u) is not a bad state in S^{N,M_N} where $S^{N,M_N} = S$ (S^{N,M_N} is the final result of the algorithm). So, (l, u) is not a blocking state in (the semantic graph of) S as (l, u) is not a bad state.

D.2.11 Proof of Theorem 5.3

We need to prove that for any other proper supervisor S': $L(S'||G) \subseteq L(S||G)$. Take arbitrary $w \in L(S'||G)$. We need to prove that $w \in L(S||G)$. Since $L(S) \subseteq L(G)$, it suffices to prove that $w \in L(S)$. We do the proof by induction on the structure of w:

Base case: Assume $w = \epsilon$. Then $w \in L(S)$ by definition.

Induction step: Assume $w = v\sigma$ for some $v \in (\Sigma_G \cup \mathbb{R}_{\geq 0})^*$ and $\sigma \in \Sigma_G \cup \mathbb{R}_{\geq 0}$ where the statement holds for v, i.e., $v \in L(S)$. It suffices to prove that the statement holds for $v\sigma$, i.e., $v\sigma \in L(S)$. To conclude that $v\sigma \in L(S)$, we prove that for all $n \leq N$, for all $m \leq M_n$: $v\sigma \in L(S^{n,m})$ using nested induction on n and m. Induction on n:

- **Base case:** $n = 0$. We prove that for all $m \leq M_0$: $v\sigma \in S^{0,m}$ by induction on m:
  - **Base case:** $v\sigma \in L(S^{0,0})$ because $S^{0,0} = G$, and $v\sigma \in L(G)$ as $v\sigma \in L(S'||G)$ by assumption.
  - **Induction step:** assume that $v\sigma \in L(S^{0,m})$. It suffices to prove that $v\sigma \in$

$L(S^{0,m+1})$. $S^{0,m+1}$ differs from $S^{0,m}$ only in terms of the guards of (some) controllable edges. So, according to Definition 5.7, $v\sigma \in L(S^{0,m+1})$ for $\sigma \in \Sigma_{uc} \cup \mathbb{R}_{\geq 0}$ since the guards of uncontrollable edges and the invariants stay the same, and already $v\sigma \in L(S^{0,m})$.

Let us say that $\sigma \in \Sigma_c$, and assume that for $v\sigma \in L(S^{0,m})$, there exists states (l_s, u_s) (in the semantic graph of $S^{0,m}$) reached by v from the initial state, and (l_t, u_t) reached from (l_s, u_s) by σ. Then, due to Definition 5.7, $u_s \models e.g^m$, and $u_s[r] \models I^0(l_t)$. To conclude that $v\sigma \in L(S^{0,m+1})$, it suffices to prove that $u_s \models e.g^{m+1}$ because the invariants stay the same. Assume that $u_s \not\models e.g^{m+1}$. Then, considering Algorithm 5.3-line 13, $u_s \models B^{0,m}(l_t)[r]$. Again by Definition 5.7, $u_s[r] = u_t$. So, $u_t \models B^{0,m}(l_t)$ as $u_s \models B^{0,m}(l_t)[r]$. Then, by Property 5.4, (l_t, u_t) is a bad state, and this contradicts the assumption that S' is a proper supervisor as it does not prevent all the bad states to take care nonblockingness and controllability.

- **Conclusion:** for all $m \leq M_0$: $v\sigma \in L(S^{0,m})$.

- **Induction step:** assume that for all $m \leq M_n$: $v\sigma \in L(S^{n,m})$. We prove that for all $m \leq M_{n+1}$: $v\sigma \in L(S^{n+1,m})$ using induction on m.
  - **Base case:** We need to prove that $v\sigma \in L(S^{n+1,0})$.

    For $v\sigma \in L(S^{n,0})$ (it holds by the induction assumption), assume that there exists some $l_s \in L$ and a clock valuation u_s such that (l_s, u_s) is reached from $(l_0, \mathbf{0})$ by v in (the semantic graph of) $S^{n,0}$. To conclude $v\sigma \in L(S^{n+1,0})$, we prove that σ occurs at (l_s, u_s) in $S^{n+1,0}$ for different cases of σ:

    σ is an event transition, related to an edge (l_s, σ, g, r, l_t). Then, according to Definition 5.7: $u_s \models e.g^0$ and $u_s[r] \models I^n_S(l_t)$. So, $u_s \models e.g^0$, and it suffices to prove that $u_s[r] \models I^{n+1}_S(l_t)$. We continue the proof by contradiction. Assume that $u_s[r] \not\models I^{n+1}_S(l_t)$. Then, based on Algorithm 5.3-line 20, $u_s[r] \models B^{n,0}(l_t)$ because already $u_s[r] \models I^n_S(l_t)$ as assumed. So, $u_t \models B^{n,0}(l_t)$ as $u_s[r] = u_t$ (again by Definition 5.7), and based on Property 5.4, (l_t, u_t) is a bad state, and this contradicts the assumption that S' is a proper supervisor.

    σ is a time transition, say Δ. Then, for $v\sigma \in L(S^{n,0})$, according to Definition 5.7: $u_s + \delta \models I^n_S(l_s)$ for all $\delta \leq \Delta$. Also, $l_t = l_s$, and $u_t = u_s + \Delta$. It suffices to prove that $u_s + \delta \models I^{n+1}_S(l_s)$ for all $\delta \leq \Delta$. By contradiction, assume that for some $\delta \leq \Delta$, $u_s + \delta \not\models I^{n+1}_S(l_s)$. Then, based on Algorithm 5.3-line 20, $u_s + \delta \models B^{n,0}(l_s)$ because $u_s + \delta \models I^{n+1}_S(l_s)$. In this case, based on Property 5.4, $(l_t, u_s + \delta)$ is a bad state. This contradicts the assumption that S' is a proper supervisor because, as a proper supervisor, it should prevent $(l_t, u_s + \delta)$. However, $(l_t, u_s + \delta)$ can be reached through the Δ transition that occurs in S'.
  - **Induction step:** assume that $v\sigma \in L(S^{n+1,m})$. Then, $v\sigma \in L(S^{n+1,m+1})$ for the same reason states in the previous induction step on m.
  - **Conclusion:** for all $m \leq M_n$: $v\sigma \in L(S^{n,m})$.

- **Conclusion:** for all $n \leq N$, for all $m \leq M_n$: $v\sigma \in L(S^{n,m})$.

Conclusion: by the principle of induction, $w \in L(S||G)$ for all $w \in L(S'||G)$.

D.2.12 Proof of Theorem 5.4

This proof is inspired from the proof of safety in [**TAC-NSC**]. Since S and G has the same event set Σ_G and $\Sigma_R \subseteq \Sigma_G$, $\Sigma_G \cap \Sigma_R = \Sigma_R$. So, it suffices to prove that if we take any $w \in P_{\Sigma_R}(L(S||(G||R^{\perp}))))$: $w \in L(R)$. Take $w \in P_{\Sigma_R}(L(S||(G||R^{\perp}))))$, then due to the projection properties, there exists $w' \in L(S||(G||R^{\perp}))$ such that $P_{\Sigma_R}(w') = w$. Also, based on Property 5.7, $L(S) \subseteq L(G||R^{\perp})$, and so $w' \in L(G||R^{\perp})$. Applying the projection on Σ_R gives $P_{\Sigma_R}(w') \in L(R^{\perp})$. For $w \in P_{\Sigma_R}(L(S||(G||R^{\perp}))) \cap L(R^{\perp})$, $w \in L(R)$ since the blocking state q_d added to $G||R$ to make $G||R^{\perp}$ is removed by S as guaranteed by Theorem 5.2.

Bibliography

Akesson, K., M. Fabian, H. Flordal, and R. Malik (2006). "Supremica-an integrated environment for verification, synthesis and simulation of discrete event systems". *Discrete Event Systems, 2006 8th International Workshop on*. IEEE, pp. 384–385.

Alur, R. (1999). "Timed automata". *International Conference on Computer Aided Verification*. Springer, pp. 8–22.

Alur, R. (2015). *Principles of cyber-physical systems*. MIT Press.

Alur, R. and D. L. Dill (1994). "A Theory of Timed Automata". *Theoretical Computer Science* vol. 126, pp. 183–235.

Alur, R., T. A. Henzinger, G. Lafferriere, and G. J. Pappas (2000). "Discrete abstractions of hybrid systems". *Proceedings of the IEEE* vol. 88, no. 7, pp. 971–984.

Alves, M. V. S., L. K. Carvalho, and J. C. Basilio (2017). "Supervisory control of timed networked discrete event systems". *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 4859–4865.

Alves, M. V. S., J. C. Basilio, A. E. C. da Cunha, L. K. Carvalho, and M. V. Moreira (2014). "Robust supervisory control against intermittent loss of observations". *IFAC Proceedings Volumes* vol. 47, no. 2, pp. 294–299.

Amin, S., X. Litrico, S. S. Sastry, and A. M. Bayen (2010). "Stealthy deception attacks on water SCADA systems". *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*. ACM, pp. 161–170.

Antsaklis, P. and J. Baillieul (2007). "Special issue on technology of networked control systems". *Proceedings of the IEEE* vol. 95, no. 1, pp. 5–8.

Asarin, E., O. Maler, A. Pnueli, and J. Sifakis (1998). "Controller synthesis for timed automata". *IFAC Proceedings Volumes* vol. 31, no. 18, pp. 447–452.

Atampore, F., J. Dingel, and K. Rudie (2016). "Automated service composition via supervisory control theory". *2016 13th International Workshop on Discrete Event Systems (WODES)*, pp. 28–35.

Baeten, J. C., J. M. van de Mortel-Fronczak, and J. E. Rooda (2016). "Integration of supervisory control synthesis in model-based systems engineering". *Complex Systems*. Springer, pp. 39–58.

Baheti, R. and H. Gill (2011). "Cyber-physical systems". *The impact of control technology* vol. 12, no. 1, pp. 161–166.

Balemi, S. (1992). "Communication delays in connections of input/output discrete event processes". *1992 31st IEEE Conference on Decision and Control*. IEEE, pp. 3374–3379.

Basile, F. and P. Chiacchio (2007). "On the implementation of supervised control of discrete event systems". *IEEE Transactions on Control Systems Technology* vol. 15, no. 4, pp. 725–739.

van Beek, D. A., W. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. Van De Mortel-Fronczak, and M. A. Reniers (2014). "CIF 3: Model-based engineering of supervisory controllers". *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, pp. 575–580.

Behrmann, G., A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime (2007). "Uppaal-tiga: Time for playing games!" *International Conference on Computer Aided Verification*. Springer, pp. 121–125.

Bengtsson, J. and W. Yi (2004). "Timed Automata: Semantics, Algorithms and Tools". *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*. Springer Berlin Heidelberg, pp. 87–124.

Bérard, B., J. Mullins, and M. Sassolas (2015). "Quantifying opacity". *Mathematical Structures in Computer Science* vol. 25, no. 2, pp. 361–403.

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

Cai, K., R. Zhang, and W. M. Wonham (2016). "Relative Observability and Coobservability of Timed Discrete-Event Systems". *IEEE Transactions on Automatic Control* vol. 61, no. 11, pp. 3382–3395.

Camtepe, S. A. and B. Yener (2007). "Modeling and detection of complex attacks". *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*. IEEE, pp. 234–243.

Cardenas, A. A., S. Amin, and S. Sastry (2008). "Secure control: Towards survivable cyber-physical systems". *Distributed Computing Systems Workshops, 2008. ICDCS'08. 28th International Conference on*. IEEE, pp. 495–500.

Cardenas, A., S. Amin, B. Sinopoli, A. Giani, A. Perrig, and S. Sastry (2009). "Challenges for securing cyber physical systems". *Workshop on future directions in cyber-physical systems security*. Vol. 5.

Carroll, J. and D. Long (1989). *Theory of Finite Automata with an Introduction to Formal Languages*. Prentice-Hall, Inc.

Carvalho, L. K., Y.-C. Wu, R. Kwong, and S. Lafortune (2018). "Detection and mitigation of classes of attacks in supervisory control systems". *Automatica* vol. 97, pp. 121–133.

Cassandras, C. G. and S. Lafortune (2009). *Introduction to discrete event systems*. Springer Science & Business Media.

Cassez, F., A. David, E. Fleury, K. G. Larsen, and D. Lime (2005). "Efficient on-the-fly algorithms for the analysis of timed games". *International Conference on Concurrency Theory*. Springer, pp. 66–80.

Cassez, F., J. Dubreil, and H. Marchand (2012). "Synthesis of opaque systems with static and dynamic masks". *Formal Methods in System Design* vol. 40, no. 1, pp. 88–115.

Ciancamerla, E., B. Fresilli, M. Minichino, T. Patriarca, and S. Iassinovski (2014). "An electrical grid and its SCADA under cyber attacks: Modelling versus a Hybrid Test Bed". *2014 International Carnahan Conference on Security Technology (ICCST)*, pp. 1–6.

Cuijpers, P. J. L., M. A. Reniers, and W. P. M. H. Heemels (2002). "Hybrid transition systems". *Technical Report. CS-Report 02-12*. Department of Computer Science, Eindhoven University of Technology.

Derler, P., E. A. Lee, and A. S. Vincentelli (2011). "Modeling cyber–physical systems". *Proceedings of the IEEE* vol. 100, no. 1, pp. 13–28.

Dubey, A. (2009). "A discussion on supervisory control theory in real-time discrete event systems". *ISIS* vol. 9, p. 112.

Dubreil, J., P. Darondeau, and H. Marchand (2010). "Supervisory Control for Opacity". *IEEE Transactions on Automatic Control* vol. 55, no. 5, pp. 1089–1100.

Ehlers, R., S. Lafortune, S. Tripakis, and M. Y. Vardi (2017). "Supervisory control and reactive synthesis: a comparative introduction". *Discrete Event Dynamic Systems* vol. 27, no. 2, pp. 209–260.

Eliades, D. G. and M. M. Polycarpou (2010). "A fault diagnosis and security framework for water systems". *IEEE Transactions on Control Systems Technology* vol. 18, no. 6, pp. 1254–1265.

Fabian, M. and A. Hellgren (1998). "PLC-based implementation of supervisory control for discrete event systems". *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*. Vol. 3. IEEE, pp. 3305–3310.

Fawzi, H., P. Tabuada, and S. Diggavi (2014). "Secure estimation and control for cyber-physical systems under adversarial attacks". *IEEE Transactions on Automatic Control* vol. 59, no. 6, pp. 1454–1467.

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

Góes, R. M., E. Kang, R. Kwong, and S. Lafortune (2017). "Stealthy deception attacks for cyber-physical systems". *Decision and Control (CDC), 2017 IEEE 56th Annual Conference on*. IEEE, pp. 4224–4230.

Gunes, V., S. Peter, T. Givargis, and F. Vahid (2014). "A survey on concepts, applications, and challenges in cyber-physical systems." *KSII Transactions on Internet & Information Systems* vol. 8, no. 12.

Gupta, R. A. and M.-Y. Chow (2010). "Networked control system: Overview and research trends". *IEEE Transactions on Industrial Electronics* vol. 57, no. 7, pp. 2527–2535.

Heemels, W. M. H., A. R. Teel, N. Van de Wouw, and D. Nesic (2010). "Networked control systems with communication constraints: Tradeoffs between transmission intervals, delays and performance". *IEEE Transactions on Automatic Control* vol. 55, no. 8, pp. 1781–1796.

Heemels, W., D. Lehmann, J. Lunze, and B. De Schutter (2009). "Introduction to hybrid systems". *Handbook of Hybrid Systems Control: Theory, Tools, Applications*. Cambridge University Press.

Hellgren, A., M. Fabian, and B. Lennartson (2001). "Modular implementation of discrete event systems as sequential function charts applied to an assembly cell". *Proceedings of the 2001 IEEE International Conference on Control Applications (CCA'01) (Cat. No.01CH37204)*, pp. 453–458.

Henzinger, T. A. (2000). "The theory of hybrid automata". *Verification of digital and hybrid systems*. Springer, pp. 265–292.

Hopcroft, J. E., R. Motwani, and J. D. Ullman (2006). *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley.

Jacob, R., J.-J. Lesage, and J.-M. Faure (2016). "Overview of discrete event systems opacity: Models, validation, and quantification". *Annual reviews in control* vol. 41, pp. 135–146.

Jech, T. (2013). *Set theory*. Springer Science & Business Media.

Jha, S., O. Sheyner, and J. Wing (2002). "Two formal analyses of attack graphs". *Computer Security Foundations Workshop, 2002. Proceedings. 15th IEEE*. IEEE, pp. 49–63.

Ji, Y., Y.-C. Wu, and S. Lafortune (2018). "Enforcement of opacity by public and private insertion functions". *Automatica* vol. 93, pp. 369–378.

Kang, E., S. Adepu, D. Jackson, and A. P. Mathur (2016). "Model-Based Security Analysis of a Water Treatment System". *2016 IEEE/ACM 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*, pp. 22–28.

Khoumsi, A. (2002). "Supervisory control of dense real-time discrete-event systems with partial observation". *Proceedings of the 6th International Workshop on Discrete Event Systems (WODES'02)*. IEEE, pp. 105–112.

Khoumsi, A. and M. Nourelfath (2002). "An efficient method for the supervisory control of dense real-time discrete event systems". *Proceedings of the 8th International Conference on Real-Time Computing Systems (RTCSA)*.

Kordy, B., L. Piètre-Cambacédès, and P. Schweitzer (2014). "DAG-based attack and defense modeling: Don't miss the forest for the attack trees". *Computer science review* vol. 13, pp. 1–38.

Koutsoukos, X. D., P. J. Antsaklis, J. A. Stiver, and M. D. Lemmon (2000). "Supervisory control of hybrid systems". *Proceedings of the IEEE* vol. 88, no. 7, pp. 1026–1049.

Lafortune, S., F. Lin, and C. N. Hadjicostis (2018). "On the history of diagnosability and opacity in discrete event systems". *Annual Reviews in Control* vol. 45, pp. 257–266.

Leal, A. B., D. L. Da Cruz, and M. d. S. Hounsell (2009). "Supervisory control implementation into programmable logic controllers". *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*. IEEE, pp. 1–7.

Lima, P. M., L. K. Carvalho, and M. V. Moreira (2018). "Detectable and Undetectable Network Attack Security of Cyber-physical Systems". *IFAC-PapersOnLine* vol. 51, no. 7, pp. 179–185.

Lima, P. M., M. V. Alves, L. K. Carvalho, and M. V. Moreira (2017). "Security Against Network Attacks in Supervisory Control Systems". *IFAC-PapersOnLine* vol. 50, no. 1, pp. 12333–12338.

Lin, F. (2011). "Opacity of discrete event systems and its applications". *Automatica* vol. 47, no. 3, pp. 496–503.

Lin, F. (2014). "Control of Networked Discrete Event Systems: Dealing with Communication Delays and Losses". *SIAM Journal on Control and Optimization* vol. 52, no. 2, pp. 1276–1298.

Lin, F. (2020). "Modeling and Control of Networked Discrete-Event Systems". Wiley Encyclopedia of Electrical and Electronics Engineering, pp. 1–27.

Lin, L., S. Thuijsman, Y. Zhu, S. Ware, R. Su, and M. Reniers (2018a). "Synthesis of Successful Actuator Attackers on Supervisors". *arXiv preprint arXiv:1807.06720*.

Lin, L., S. Thuijsman, Y. Zhu, S. Ware, R. Su, and M. Reniers (2018b). "Synthesis of Supremal Successful Normal Actuator Attackers on Normal Supervisors". *American Control Conference*.

Liu, Z., X. Yin, S. Shu, and S. Li (2019). "Online Supervisory Control of Networked Discrete-Event Systems with Control Delays". *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, pp. 6706–6711.

Maler, O., A. Pnueli, and J. Sifakis (1995). "On the synthesis of discrete controllers for timed systems". *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, pp. 229–242.

Malik, P. (2002). "Generating controllers from discrete-event models".

Meira-Góes, R., E. Kang, R. H. Kwong, and S. Lafortune (2020). "Synthesis of sensor deception attacks at the supervisory layer of Cyber–Physical Systems". *Automatica* vol. 121, p. 109172.

Miao, C., S. Shu, and F. Lin (2019). "Predictive Supervisory Control for Timed Discrete Event Systems under Communication Delays". *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, pp. 6724–6729.

Miremadi, S., Z. Fei, K. Åkesson, and B. Lennartson (2015). "Symbolic supervisory control of timed discrete event systems". *IEEE Transactions on Control Systems Technology* vol. 23, no. 2, pp. 584–597.

Moor, T. (2016). "A discussion of fault-tolerant supervisory control in terms of formal languages". *Annual Reviews in Control* vol. 41, pp. 159–169.

Nicolaou, N., D. G. Eliades, C. Panayiotou, and M. M. Polycarpou (2018). "Reducing Vulnerability to Cyber-Physical Attacks in Water Distribution Networks". *2018 International Workshop on Cyber-physical Systems for Smart Water Networks (CySWater)*. IEEE, pp. 16–19.

Ostroff, J. S. (1990). "Deciding properties of timed transition models". *IEEE Transactions on Parallel and Distributed Systems* vol. 1, no. 2, pp. 170–183.

Ouedraogo, L., M. N. El Fath, and A. Khoumsi (2008). *Setexp: A method of transformation of timed automata into finite state automata*. CIRRELT.

Ouedraogo, L., A. Khoumsi, and M. Nourelfath (2010). "SetExp: a method of transformation of timed automata into finite state automata". *Real-Time Systems* vol. 46, no. 2, pp. 189–250.

Ouedraogo, L., R. Kumar, R. Malik, and K. Akesson (2011). "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.

Paoli, A. and S. Lafortune (2005). "Safe diagnosability for fault-tolerant supervision of discrete-event systems". *Automatica* vol. 41, no. 8, pp. 1335–1347.

Paoli, A., M. Sartini, and S. Lafortune (2011). "Active fault tolerant control of discrete event systems using online diagnostics". *Automatica* vol. 47, no. 4, pp. 639–649.

Park, S.-J. (2012). "Robust and nonblocking supervisory control of nondeterministic discrete event systems with communication delay and partial observation". *International journal of control* vol. 85, no. 1, pp. 58–68.

Park, S.-J. and K.-H. Cho (2006). "Delay-robust supervisory control of discrete-event systems with bounded communication delays". *IEEE Transactions on Automatic Control* vol. 51, no. 5, pp. 911–915.

Park, S.-J. and K.-H. Cho (2008). "Nonblocking supervisory control of timed discrete event systems under communication delays: The existence conditions". *Automatica* vol. 44, no. 4, pp. 1011–1019.

Pasqualetti, F., F. Dörfler, and F. Bullo (2013). "Attack Detection and Identification in Cyber-Physical Systems". *IEEE Transactions on Automatic Control* vol. 58, no. 11, pp. 2715–2729.

Pasqualetti, F., F. Dörfler, and F. Bullo (2011). "Cyber-physical attacks in power networks: Models, fundamental limitations and monitor design". *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*. IEEE, pp. 2195–2201.

Prenzel, L. and J. Provost (2018). "PLC implementation of symbolic, modular supervisory controllers". *IFAC-PapersOnLine* vol. 51, no. 7, pp. 304–309.

van Putten, B. J. C., B. van der Sanden, M. Reniers, J. Voeten, and R. Schiffelers (2020). "Supervisor synthesis and throughput optimization of partially-controllable manufacturing systems". *Discrete Event Dynamic Systems*, pp. 1–33.

Ramadge, P. J. and W. M. Wonham (1987). "Supervisory control of a class of discrete event processes". *SIAM Journal on Control and Optimization* vol. 25, no. 1, pp. 206–230.

Ramadge, P. and W. Wonham (1984). "Supervisory control of a class of discrete event processes". *Analysis and Optimization of Systems*. Springer, pp. 475–498.

Rashidinejad, A., P. van der Graaf, and M. Reniers (2020a). "Nonblocking Supervisory Control Synthesis of Timed Automata using Abstractions and Forcible Events". *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE, pp. 1–8.

Rashidinejad, A., P. van der Graaf, M. Reniers, and M. Fabian (2020b). "Non-blocking Supervisory Control of Timed Automata Using Forcible Events". *15th International Workshop on Discrete Event Systems (WODES 2020)*. Accepted. IEEE.

Rashidinejad, A., M. Reniers, and M. Fabian (2019a). "Supervisory control of discrete-event systems in an asynchronous setting". *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*. IEEE, pp. 494–501.

Rashidinejad, A., M. Reniers, and M. Fabian (2021a). "Networked Supervisory Control Synthesis of Timed Discrete-Event Systems". *arXiv preprint arXiv 2102.09255*.

Rashidinejad, A., M. Reniers, and M. Fabian (2021b). "Supervisory Control Synthesis of Timed Automata Using Forcible Events". *arXiv preprint arXiv 2102.09338*.

Rashidinejad, A., M. Reniers, and L. Feng (2018). "Supervisory Control of Timed Discrete-Event Systems Subject to Communication Delays and Non-FIFO Observations". *IFAC-PapersOnLine* vol. 51, no. 7. 14th IFAC Workshop on Discrete Event Systems WODES 2018, pp. 456–463.

Rashidinejad, A., B. Wetzels, M. Reniers, L. Lin, Y. Zhu, and R. Su (2019b). "Supervisory control of discrete-event systems under attacks: an overview and outlook". *2019 18th European Control Conference (ECC)*. IEEE, pp. 1732–1739.

Reijnen, F. F., M. A. Goorden, J. M. van de Mortel-Fronczak, and J. E. Rooda (2020). "Modeling for supervisor synthesis–a lock-bridge combination case study". *Discrete Event Dynamic Systems* vol. 30, no. 3, pp. 499–532.

Reniers, M., J. van de Mortel-Fronczak, and K. Roelofs (2017). "Model-based engineering of supervisory controllers for cyber-physical systems". *Industrial Internet of Things*. Springer, pp. 111–136.

Rohloff, K. (2012). "Bounded sensor failure tolerant supervisory control." *WODES*, pp. 272–277.

Saboori, A. and C. N. Hadjicostis (2007). "Notions of security and opacity in discrete event systems". *Decision and Control, 2007 46th IEEE Conference on*. IEEE, pp. 5056–5061.

Saboori, A. and C. N. Hadjicostis (2012). "Opacity-enforcing supervisory strategies via state estimator constructions". *IEEE Transactions on Automatic Control* vol. 57, no. 5, pp. 1155–1165.

Sha, L., S. Gopalakrishnan, X. Liu, and Q. Wang (2008). "Cyber-physical systems: A new frontier". *2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (sutc 2008)*. IEEE, pp. 1–9.

Shi, J., J. Wan, H. Yan, and H. Suo (2011). "A survey of cyber-physical systems". *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*. IEEE, pp. 1–6.

Shu, S. and F. Lin (2014). "Fault-Tolerant Control for Safety of Discrete-Event Systems". *IEEE Transactions on Automation Science and Engineering* vol. 11, no. 1, pp. 78–89.

Shu, S. and F. Lin (2015). "Supervisor synthesis for networked discrete event systems with communication delays". *IEEE Transactions on Automatic Control* vol. 60, no. 8, pp. 2183–2188.

Shu, S. and F. Lin (2017a). "Deterministic Networked Control of Discrete Event Systems with Nondeterministic Communication Delays". *IEEE Transactions on Automatic Control* vol. 62, no. 1, pp. 190–205.

Shu, S. and F. Lin (2017b). "Predictive Networked Control of Discrete Event Systems". *IEEE Transactions on Automatic Control* vol. 62, no. 9, pp. 4698–4705.

Skoldstam, M., K. Akesson, and M. Fabian (2007). "Modeling of discrete event systems using finite automata with variables". *2007 46th IEEE Conference on Decision and Control*. IEEE, pp. 3387–3392.

Sridhar, S., A. Hahn, and M. Govindarasu (2011). "Cyber–physical system security for the electric power grid". *Proceedings of the IEEE* vol. 100, no. 1, pp. 210–224.

Su, R. (2018). "Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations". *Automatica* vol. 94, pp. 35–44.

Svoreňová, M. and M. Kwiatkowska (2016). "Quantitative verification and strategy synthesis for stochastic games". *European Journal of Control* vol. 30, pp. 15–30.

Swartjes, L., D. van Beek, W. J. Fokkink, and J. van Eekelen (2017). "Model-based design of supervisory controllers for baggage handling systems". *Simulation Modelling Practice and Theory* vol. 78, pp. 28–50.

Takai, S. and T. Ushio (2006). "A new class of supervisors for timed discrete event systems under partial observation". *Discrete Event Dynamic Systems* vol. 16, no. 2, pp. 257–278.

Teixeira, A., H. Sandberg, and K. H. Johansson (2010). "Networked control systems under cyber attacks with applications to power networks". *Proceedings of the 2010 American Control Conference*, pp. 3690–3696.

Teixeira, A., D. Pérez, H. Sandberg, and K. H. Johansson (2012). "Attack Models and Scenarios for Networked Control Systems". *Proceedings of the 1st International Conference on High Confidence Networked Systems*. Beijing, China: ACM, pp. 55–64.

Theunissen, R. J., M. Petreczky, R. R. Schiffelers, D. A. van Beek, and J. E. Rooda (2013). "Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner". *IEEE Transactions on Automation Science and Engineering* vol. 11, no. 1, pp. 20–32.

Thoben, K.-D., S. Wiesner, and T. Wuest (2017). ""Industrie 4.0" and smart manufacturing-a review of research issues and application examples". *International journal of automation technology* vol. 11, no. 1, pp. 4–16.

Thorsley, D. and D. Teneketzis (2006). "Intrusion detection in controlled discrete event systems". *Decision and Control, 2006 45th IEEE Conference on*. IEEE, pp. 6047–6054.

Tiwari, A. (2008). "Abstractions for hybrid systems". *Formal Methods in System Design* vol. 32, no. 1, pp. 57–83.

Tong, Y., K. Cai, and A. Giua (2018a). "Decentralized Opacity Enforcement in Discrete Event Systems Using Supervisory Control". *2018 57th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pp. 1053–1058.

Tong, Y., Z. Li, C. Seatzu, and A. Giua (2018b). "Current-state opacity enforcement in discrete event systems under incomparable observations". *Discrete Event Dynamic Systems* vol. 28, no. 2, pp. 161–182.

Tripakis, S. and K. Altisen (1999). "On-the-fly controller synthesis for discrete and dense-time systems". *International Symposium on Formal Methods*. Springer, pp. 233–252.

Tripakis, S. and S. Yovine (2001). "Analysis of timed systems using time-abstracting bisimulations". *Formal Methods in System Design* vol. 18, no. 1, pp. 25–68.

Uma, M. and G. Padmavathi (2013). "A Survey on Various Cyber Attacks and their Classification." *IJ Network Security* vol. 15, no. 5, pp. 390–396.

Ushio, T. and S. Takai (2016). "Nonblocking supervisory control of discrete event systems modeled by mealy automata with nondeterministic output functions". *IEEE Transactions on Automatic Control* vol. 61, no. 3, pp. 799–804.

Wakaiki, M., P. Tabuada, and J. P. Hespanha (2017). "Supervisory control of discrete-event systems under attacks". *Dynamic Games and Applications*, pp. 1–19.

Ware, S. and R. Malik (2008). "The Use of language projection for compositional verification of discrete event systems". *2008 9th International Workshop on Discrete Event Systems*. IEEE, pp. 322–327.

Wittmann, T., J. Richter, and T. Moor (2012). "Fault-tolerant control of discrete event systems based on fault-accommodating models". *IFAC Proceedings Volumes* vol. 45, no. 20, pp. 854–859.

Wong-Toi, H. and G. Hoffmann (1991). "The control of dense real-time discrete event systems". *Proceedings of the 30th IEEE Conference on Decision and Control*, pp. 1527–1528.

Wonham, W. M. (2015). "Supervisory control of discrete-event systems". *Encyclopedia of Systems and Control*, pp. 1396–1404.

Wonham, W. M. and K. Cai (n.d.). *Supervisory control of discrete-event systems*. Springer.

Wu, B., J. Dai, and H. Lin (2018). "Synthesis of insertion functions to enforce decentralized and joint opacity properties of discrete-event systems". *2018 Annual American Control Conference (ACC)*. IEEE, pp. 3026–3031.

Wu, Y.-C. and S. Lafortune (2014). "Synthesis of insertion functions for enforcement of opacity security properties". *Automatica* vol. 50, no. 5, pp. 1336–1348.

Xu, P., S. Shu, and F. Lin (2017). "Nonblocking networked control of discrete event systems". *2017 Chinese Automation Congress (CAC)*, pp. 1911–1916.

Xu, S. and R. Kumar (2008). "Asynchronous implementation of synchronous discrete event control". *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*. IEEE, pp. 181–186.

Yao, J., X. Yin, and S. Li (2020). "On Attack Mitigation in Supervisory Control Systems: A Tolerant Control Approach". *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, pp. 4504–4510.

Yin, X. (2017). "Supervisor synthesis for mealy automata with output functions: A model transformation approach". *IEEE Transactions on Automatic Control* vol. 62, no. 5, pp. 2576–2581.

Yin, X. and S. Lafortune (2015). "Synthesis of maximally permissive supervisors for partially-observed discrete-event systems". *IEEE Transactions on Automatic Control* vol. 61, no. 5, pp. 1239–1254.

Yin, X. and S. Lafortune (2016). "A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems". *IEEE Transactions on Automatic Control* vol. 61, no. 8, pp. 2140–2154.

Yin, X. and S. Li (2018a). "Synthesis of dynamic masks for infinite-step opacity". *IFAC-PapersOnLine* vol. 51, no. 7, pp. 343–348.

Yin, X. and S. Li (2018b). "Verification of opacity in networked supervisory control systems with insecure control channels". *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, pp. 4851–4856.

Zaytoon, J. and B. Riera (2017). "Synthesis and implementation of logic controllers–a review". *Annual reviews in control* vol. 43, pp. 152–168.

Zhang, Q., Z. Li, C. Seatzu, and A. Giua (2018). "Stealthy Attacks for Partially-Observed Discrete Event Systems". *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. IEEE, pp. 1161–1164.

Zhao, B., F. Lin, C. Wang, X. Zhang, M. P. Polis, and L. Y. Wang (2017). "Supervisory control of networked timed discrete event systems and its applications to power distribution networks". *IEEE Transactions on Control of Network Systems* vol. 4, no. 2, pp. 146–158.

Zhu, Q. and T. Başar (2011). "Robust and resilient control design for cyber-physical systems with an application to power systems". *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*. IEEE, pp. 4066–4071.

Zhu, Y., L. Lin, and R. Su (2019a). "Supervisor obfuscation against actuator enablement attack". *2019 18th European Control Conference (ECC)*. IEEE, pp. 1760–1765.

Zhu, Y., L. Lin, S. Ware, and R. Su (2019b). "Supervisor synthesis for networked discrete event systems with communication delays and lossy channels". *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, pp. 6730–6735.

Acknowledgments

Gaining a Ph.D. degree is not solely about growing knowledge; it is a masterclass of personal development. I would like to thank the following people, whose presence in my life made this journey happen.

My first promoter, Michel, words cannot express how thankful I am for your great supervision and guidance through each step of this process. You patiently helped me to not only enhance my knowledge but also my interpersonal skills. You always supported and trusted me and my ideas. This thesis involves a lot of analytical proofs, which looked almost impossible to have. Our meetings sometimes lasted three hours to eventually make them possible.

My second promoter, Martin, I was thrilled when you accepted to become my supervisor, and as time passed, I started to feel more and more proud about it. I know I asked for comments many times, but I did not expect to be surprised each time by the number of detailed and to-the-point comments that I received from you. I cannot thank you enough for the time and effort you dedicated to improving the quality of this work.

Other committee members, Christoforos, Jeroen, Wan, I truly appreciate the time and effort you dedicated to reviewing my thesis and all the positive feedback.

From the CST group, Maurice and Maarten, I am grateful for providing me the opportunity to experience teaching at TU/e. Maurice, you were my first promoter for a short time in the beginning. But yet it was amazing, and I was always excited for our meetings. Thomas, Martijn, Ferdie, Joshua, with whom I shared office the longest, you are the most hardworking people I ever know, and this always inspired me to work harder. Roy, I enjoyed our coffee chats during the time that we shared office. My talented students, Pepijn, Bart, Sander, and Patrick, it was my pleasure to be your supervisor; with special thanks to Patrick whose initial effort helped me create two interesting chapters of this thesis. Albert, thank you for putting effort into implementing our technique in CIF. Petra, Nancy, Roos, and Geertje, you were always so kind to me. I enjoyed any moment of chatting with you.

I was among 15 Ph.D. students involved in the oCPS project. I thank the organizers for arranging meetings, courses, internships, and summer schools. All those experiences allowed us to grow our knowledge, collaboration, communication, and presentation skills, and of course, we traveled a lot and had a lot of fun; with special thanks to Asad, Hadi, Tahira, and Precious for all the fun chats during the breaks. Lei, I am grateful for our in-depth discussions during my visit to KTH, which resulted in a great publication. Dragan,

although I had a short visit to Siemens, our meetings gave me the opportunity to get inspired and realize how my research can be beneficial in practice. Rong Su and Liyong Lin from Nanyang University, Singapore, it might be challenging to collaborate with you regarding the time difference, but yet very fruitful.

Needless to say that my friends had an important role in improving my work-life balance in this journey. I thank them all and especially the following ones. Samaneh, I specifically appreciate your kind support in the first months of this journey; you highly helped me to control the feeling of migration. Marzieh, thank you for being there with me any time I needed to share my feelings. Whenever I felt happy, sad, angry, or anxious, I knew that I have you to talk to, to cheer, to cry, and to chill together. Yasaman, thank you for always encouraging me to find time to dance. Arezoo, many times we had a heavy workload, but yet we managed to have our coffee chats. My cousin, Farnoosh, it feels great to have you in the Netherlands. With you, I feel at home, especially when we talk about our sweet childhood memories. Tahsin, I never forget the taste of the delicious meals you made for me. Sina, thank you for making coffee and lunch breaks so fun and memorable. My friends since doing a bachelor's degree, Elham, Yasaman, and Zahra (just the 4 of us), my adulthood is full of nice memories with you; with special thanks to Zahra, you have always supported me, no matter how far I am. My lovely neighbors, thank you for all the support, specifically during the lockdown. Kaveh, gardash, you make the weekends great by bringing so much joy, positive energy, and wonderful gifts with you. Mrs. and Mr. Samadikhah, you are always so kind to me. Every time I talk to you, my soul becomes full of happiness and love. My dearest Pouya, you are the most influential person in my life and my today's success. You always believed in me and encouraged me to keep going, be strong (Coach Kozak and Claudia: you are a fighter, not a quitter), and ambitious. I cannot thank you enough for all your love, support, and caring during this time. You are the one, who always makes all the happy moments memorable, and in all the difficult moments, you not only feel my problems as they are yours but also hold my hands tight until I overcome them.

Definitely, my family had the highest impact on who I am today. My sister, Atena, you always encouraged me to study hard since my childhood, and I believe it is time to see the result. My niece, Hanita, you are the youngest but yet one of the most important members of my support team. Every time I feel low, it is enough to picture your cute face to boost my mood. My father, Hossein, you have always supported me in every stage of my life, in whatever aspect and at any age. I vividly remember the way you taught me math and art; you transferred your passion to me and made them mine. My mother, Narges, you are the kindest and the most patient person that I ever know. I learned how to be grateful, be patient, make friends and be kind to people, handle problems, and even be a good cook from you. Maman and Baba, you encouraged me to go and grow, while trying to deal with my absence. I dedicate this thesis to you with love.

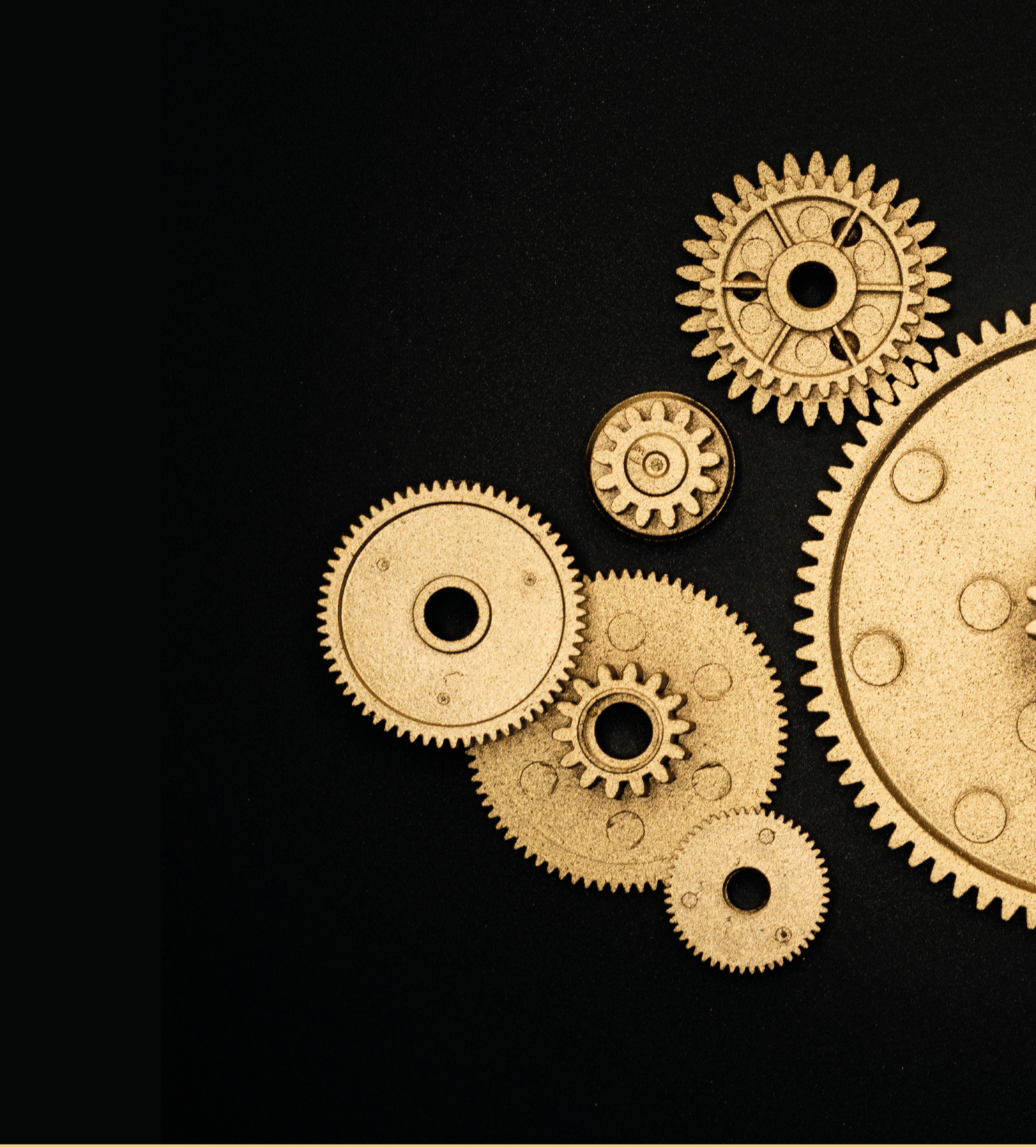
Aida Rashidinejad
April, 2021

About the Author

Aida Rashidinejad was born on July 20, 1989, in Tehran, Iran. She received her Bachelor of Science degree (graduated with honors) in electrical engineering with a specialization in control engineering from Iran University of Science and Technology (IUST), Tehran, Iran, in 2011. She received her Master of Science degree in electrical engineering with a specialization in control engineering from Amirkabir University of Technology (AUT), Tehran, Iran, in 2014.

In July 2016, she started to pursue a Ph.D. degree within the Control Systems Technology group at the Department of Mechanical Engineering at Eindhoven University of Technology (TU/e), under the supervision of Michel Reniers from TU/e and Martin Fabian from Chalmers University of Technology, Gothenburg, Sweden. Her research project was part of a research program "optimization of Cyber-Physical Systems (oCPS)" funded from the European Union's Horizon 2020 Framework Program and was focused on the model-based supervisory control of cyber-physical systems. The main results of her research are printed in this dissertation. During her Ph.D., Aida was a visiting researcher at KTH Royal Institute of Technology, Stockholm, Sweden, and at Siemens Corporate Technology, Munich, Germany.

It's all about time!



TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY