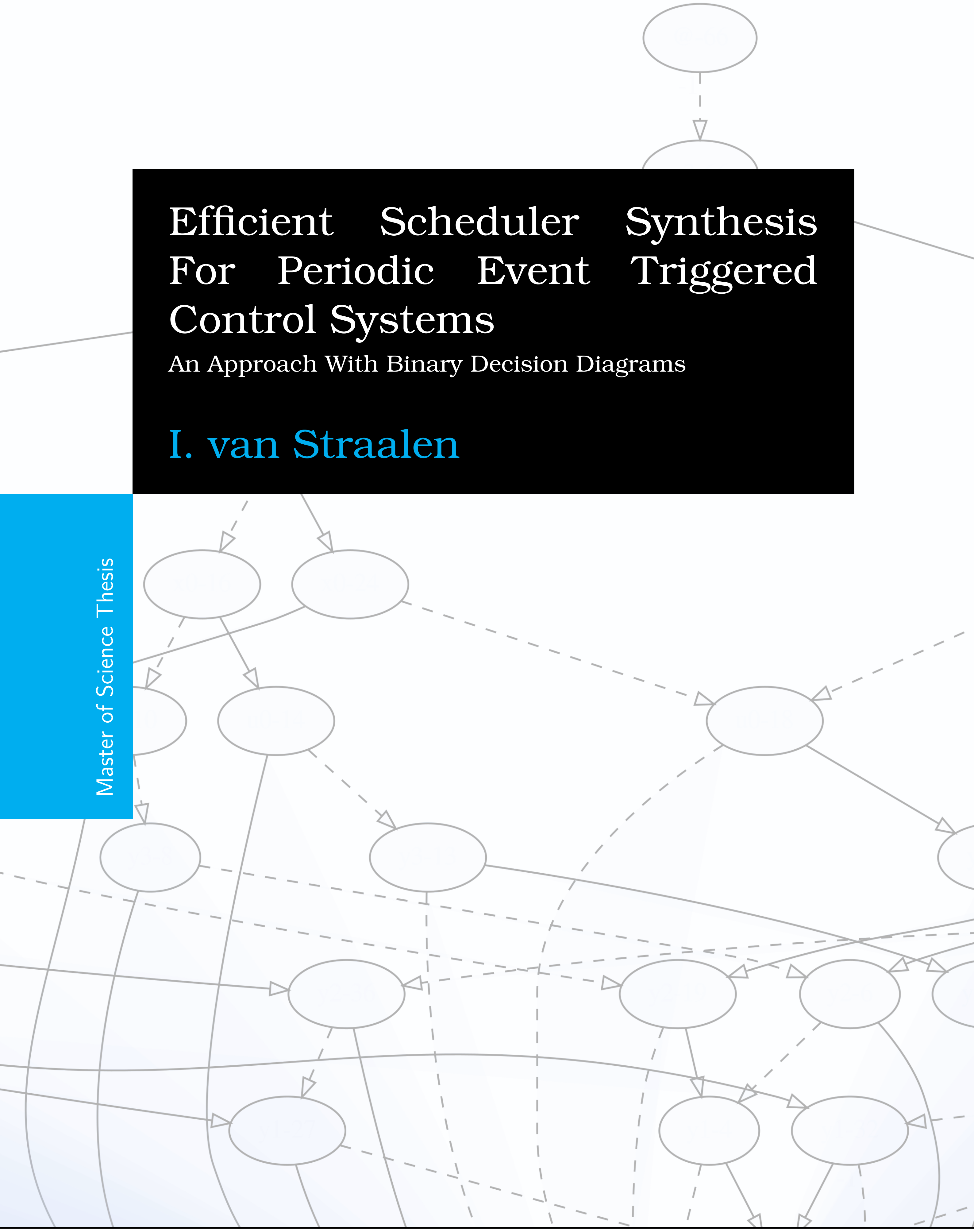


Efficient Scheduler Synthesis For Periodic Event Triggered Control Systems

An Approach With Binary Decision Diagrams

I. van Straalen

Master of Science Thesis

Delft University of Technology

Delft Center for Systems and Control

Efficient Scheduler Synthesis For Periodic Event Triggered Control Systems

An Approach With Binary Decision Diagrams

Master of Science Thesis

For the degree of Master of Science in Embedded Systems at Delft University of Technology

I. van Straalen

August 11, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

In recent years, Networked Control Systems (NCS) have become more popular, partly because of the increasing accessibility. In NCS, multiple plants and controllers are connected over a wired or wireless shared network, possibly having significant spatial separation. A major issue that arises is network congestion: If too many control loops are connected to the network, the shared communication channels become oversaturated, causing the packages to be lost, and subsequently the individual control loops might become unstable.

One solution to this problem is to make use of Periodic Event Triggered Control (PETC), where a triggering condition is checked periodically, and if this condition is satisfied the control loop is closed. Control using PETC becomes inherently aperiodic, as opposed to the periodic nature of standard control implementation. This aperiodicity introduces another issue: Avoiding collision of communication events caused by the triggering of control loops.

To resolve this issue, schedulers have to be designed for the control loops. There are approaches already available that can automatically synthesize schedulers for a collection of PETC systems. However, these share a common issue in that these scale poorly with the number of subsystems.

This thesis explores new algorithms for synthesization of schedulers for PETC, with the goal of better scalability. This is done by first abstracting the triggering behaviour of the individual control loops and representing these by Transition System (TS). Then schedulers are synthesized by solving a safety game. To increase efficiency of the safety game, several states are combined by partitioning. Additionally, a major boost in performance is gained by representing the TSs by Binary Decision Diagrams (BDDs). Finally, a method to increase schedulability is also investigated by allowing the control loops to occasionally trigger late.

Table of Contents

List of Figures

Acknowledgements and Preface

I would like to thank my supervisor dr.ir. M. Mazo Espinosa for their assistance and guidance during the writing of this thesis. Furthermore, I would like to thank every member of the research group for the discussion and insights during the meetings which helped me progress further.

This work is supported by the ERC Starting Grant SENTIENT 755953.

Delft, University of Technology I. van Straalen
August 11, 2021

Chapter 1

Introduction

In modern control theory, the control action is typically (digitally) implemented in a periodic manner, i.e. every period the current state is measured or estimated, a suitable control action is computed and finally it is actuated. A disadvantage of this approach is that it can be quite inefficient in terms of energy consumption, communication and computation as new control actions are computed and transmitted even when not necessary to sustain stability. These do generally not pose large constraints in regular systems, however, in Networked Control Systems (NCS) [2], these issues are magnified. In NCS, multiple plants and controllers, which are spatially separated, are connected by a common network. An illustration of this is shown in Figure 1-1. The interest in NCS has increased greatly in the last few years, largely because (wireless) networks have become more readily available. In the case of NCS, communication especially poses tight constraints on the systems, as often only 1 or a small amount of (bandwidth-limited) communication channels are shared by multiple control loops.

A common way to alleviate this issue is by reducing the amount of communications needed for each control loop by utilising event-based control methods. These are inherently aperiodic and only send control actions to the plant when a certain event happens. These events can be designed in such a way that the amount of communication that is needed is possibly reduced by only sending when absolutely necessary. One way to design such an event-generator is by using a triggering condition. This called Event Triggered Control (ETC). In ETC, 'out of date' versions of the state are used to calculate the control action. Only when the triggering condition is satisfied, is the current state transmitted to the controller. Of course the triggering condition has to be designed in such a way that the system remains stable. This triggering condition has to be checked continuously, which is practically impossible, so often it is checked with a small period to simulate it. It is possible to convert the triggering condition to one that is checked with a period h, that also takes into account what happens in between the samples. This triggering condition will then also trigger if the original triggering condition would have triggered during the next period. This approach is called Periodic Event Triggered Control (PETC).

One issue that PETC does not resolve is the conflicts that can arise between different control loops that would like to trigger simultaneously. A solution to this problem is to design a

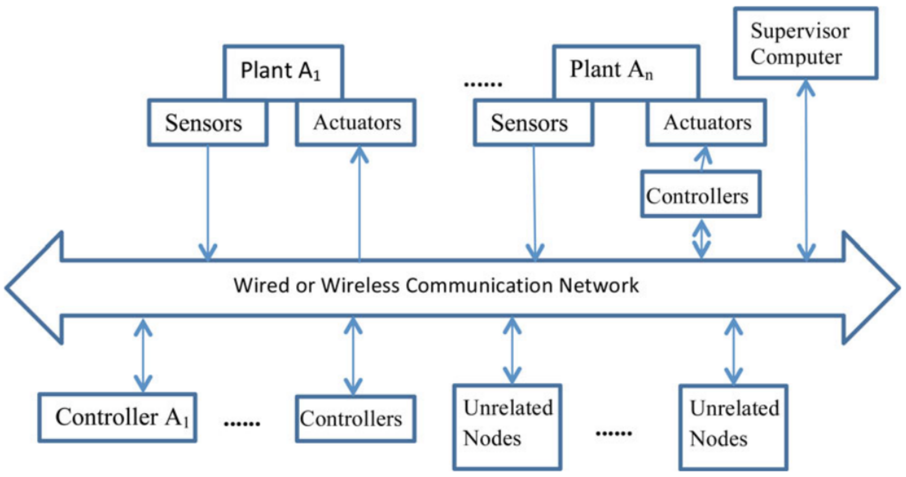


Figure 1-1: A depiction of an NCS, including multiple controllers and plants and other unrelated nodes [1].

scheduler that decides which control loops should trigger at the current sample. To make the scheduling problem more feasible, the control loops should be allowed to trigger early, i.e. before their triggering condition is satisfied. The scheduler can then ensure stability of all the control loops if they trigger before or at the time instant the triggering condition is first satisfied. Algorithms exists to synthesize schedulers for PETC systems [3, 4, 5] which make use of Timed Automata, but they do not scale very well. The goal of this thesis is thus then to design and implement more efficient and better scaling algorithms that automatically synthesize a scheduler for a given set of PETC systems.

First, traffic models which capture the triggering behaviour of the PETC systems are constructed using the techniques from [6]. These models are then converted into a particular form before they are combined, and used in a safety game. The solution to this safety game is then used to construct the scheduler. More efficient algorithms are investigated by combining several states in these models or by representing them by Binary Decision Diagrams (BDDs).

1-1 Notation

$\mathbb{R}$ represents the set of real numbers, $\mathbb{N}$ represents the set of natural numbers and $\mathbb{B}$ represents the set of boolean numbers, i.e. $\mathbb{B} = \{0, 1\}$. For a set A, $|A|$ denotes its cardinality and for a vector ξ, $|\xi|$ denotes its 2-norm. A relation Q is a subset of the product of two sets: $Q \subseteq A \times B$. Denote by $\pi_Q(a) := \{b \in B | \, (a, b) \in Q\} \subseteq B$ the set of states related to a via Q, and π_Q^{-1} defined inversely. If $A = B$, Q is called an equivalence relation, and their elements are grouped into equivalence classes, denoted by $[a]_Q := \pi_Q(a) \subseteq A$. For an arbitrary dynamical system $\dot{\xi}(t) = f(\xi(t))$, $\xi(t) \in \mathbb{R}^n$, denote the solution with initial condition $\xi(0) = x$ as $\xi_x(t)$.

1-2 Organization

The contents of this thesis are organized as follows:

- Chapter 2 contains preliminary theory required for the remainder of the thesis,
- Chapter 3 discusses how the acquired traffic models are converted as well as the main synthesization algorithms that are performed on these,
- Chapter 4 covers how the models are represented using BDDs, and how the synthesization algorithms can be expressed using BDDs,
- Chapter 5 discusses how some schedulers can be found for unschedulable systems by allowing some late triggers,
- Chapter 6 covers shortly how the algorithms are implemented, and make some considerations on how the resulting schedulers would be implemented physically,
- Chapter 7 contains some benchmarks of the algorithms and simulations using the resulting schedulers, and
- Chapter 8 concludes the thesis and makes some comments about possible future work.

Chapter 2

Preliminaries

2-1 Transition Systems

Transition systems are abstract systems consisting of states (possibly infinite) and transitions between states. These can be used to model a large variety of systems. In this thesis they are used to model triggering behaviour of Periodic Event Triggered Control (PETC) systems.

Following are some definitions and theorems used later on for the abstraction and scheduling algorithms. The notation and names in the definitions are slightly modified in some cases for consistent notation and naming conventions.

Definition 2-1.1 (Transition System (TS) [7]). *A Transition System S is a sextuple $(X, X_0, U, \longrightarrow, Y, H)$, where:*

- *a set of states X;*
- *a set of initial states $X_0 \subseteq X$;*
- *a set of inputs U;*
- *a transition relation $\longrightarrow \subseteq X \times U \times X$;*
- *a set of outputs Y;*
- *an output map $H\colon X \to Y$.*

From the definition of a TS, three additional useful sets can be defined: First the set of successors of x with action u:

$$\begin{aligned} Post_u(x) &:= \{x' \in X \,|\, \exists (x, u, x') \in \longrightarrow\}, \text{and} \\ Post(x) &:= \bigcup_{u \in U} Post_u(x). \end{aligned} \tag{2-1}$$

For some states, $Post_u(x)$ might be empty, thus denote by $U(x)$ the set of actions for which $Post_u(x)$ is nonempty:

$$U(x) := \{u \in U |\, Post_u(x) \neq \emptyset\}. \tag{2-2}$$

Finally, the set of predecessors of x:

$$Pre(x) := \{x' \in X |\, \exists (x', u, x) \in \longrightarrow\}. \tag{2-3}$$

When considering sets of states, these sets resolve to be simply the union over the individual members, i.e.:

$$Post(C) = \bigcup_{x \in C} Post(x). \tag{2-4}$$

Definition 2-1.2 (Composition of Systems [7])**.** *Let $S_a = (X_a, X_{0a}, U_a, \xrightarrow[a]{}, U_a, H_a)$ and $S_b = (X_b, X_{0b}, Y_b, \xrightarrow[b]{}, Y_b, H_b)$ be two systems and let $\mathcal{I} \subseteq X_a \times X_b \times U_a \times U_b$ be a relation. The composition of S_a and S_b with interconnection relation $\mathcal{I}$, denoted by $S_a \times_{\mathcal{I}} S_b$, is the system $(X_{ab}, X_{0ab}, U_{ab}, \xrightarrow[ab]{}, Y_{ab}, H_{ab})$, consisting of:*

- $X_{ab} = \pi_X(\mathcal{I})$;
- $X_{ab0} = X_{ab} \cap (X_{a0} \times X_{b0})$;
- $U_{ab} = U_a \times U_b$;
- $(x_a, x_b) \xrightarrow[ab]{(u_a, u_b)} (x'_a, x'_b)$ *if the following conditions hold:*
    1. $x_a \xrightarrow[a]{u_a} x'_a$ *in* S_a;
    2. $x_b \xrightarrow[b]{u_b} x'_b$ *in* S_b;
    3. $(x_a, x_b, u_a, u_b) \in \mathcal{I}$;
- $Y_{ab} = Y_a \times Y_b$;
- $H_{ab}(x_a, x_b) = (H_a(x_a), H_b(x_b))$.

Definition 2-1.3 (Simulation Relation [7])**.** *Consider two systems S_a and S_b with $Y_a = Y_b$. A relation $R \subseteq X_a \times X_b$ is a simulation relation from S_a to S_b iff:*

1. $\forall x_{a0} \in X_{a0} : \exists x_{b0} \in X_{b0}$ *with* $(x_{a0}, x_b0) \in R$;
2. $\forall (x_a, x_b) \in R : H_a(x_a) = H_b(x_b)$;
3. $\forall (x_a, x_b) \in R$ *it holds that* $\forall (x_a \xrightarrow[a]{u_a} x'_a)$ *in* S_a *implies the existence of* $(x_b \xrightarrow[b]{u_b} x'_b)$ *in* S_b *such that* $(x'_a, x'_b) \in R$.

If such a relation exists, it is said that S_a is simulated by S_b, denoted by $S_a \preceq_{\mathcal{S}} S_b$.

Definition 2-1.4 (Bisimulation [7])**.** *Given two systems S_a and S_b with $Y_a = Y_b$, S_a is bisimilar to S_b, denoted by $S_a \cong_{\mathcal{S}} S_b$, if there exists a relation R satisfying:*

1. *R is a simulation relation from S_a to S_b;*
2. *R^{-1} is a simulation relation from S_b to S_a.*

Definition 2-1.5 (Quotient System [7]). *Let $S = (X, X_0, U, \longrightarrow, Y, H)$ be a system and let Q be an equivalence relation on X such that $(x, x') \in Q$ implies $H(x) = H(x')$. The quotient of S by Q, denoted by $S_{/Q}$, is the system $(X_{/Q}, X_{/Q0}, U_{/Q}, \underset{/Q}{\longrightarrow}, Y_{/Q}, H_{/Q})$, where*

- *$X_{/Q} = X/Q$;*
- *$X_{/Q0} = \{x_{/Q0} \in X_{/Q0} | x_{/Q0} \cap X_0 \neq \emptyset\}$;*
- *$U_{/Q} = U$;*
- *$x_{/Q} \underset{/Q}{\xrightarrow{u}} x'_{/Q}$ if there exists $x \xrightarrow{u} x'$ in S with $x \in x_{/Q}$ and $x' \in x'_{/Q}$;*
- *$Y_{/Q} = Y$;*
- *$H_{/Q}(x_{/Q}) = H(x)$ for some $x \in x_{/Q}$.*

This is sometimes also called a symbolic model of S since each state $x_{/Q} \in X_{/Q}$ can be regarded as a symbol representing all the states $\pi_Q^{-1}(x_{/Q}) \subseteq X$.

Theorem 2-1.1 ([7]). *$S = (X, X_0, U, \longrightarrow, Y, H)$ be a system and let Q be an equivalence relation on X such that $(x, x') \in Q$ implies $H(x) = H(x')$. The relation:*

$$\Gamma(\pi_Q) = \{(x, x_{/Q}) \in X \times X_{/Q} | \, x_{/Q} = \pi_Q(x)\} \tag{2-5}$$

is a simulation relation from S to $S_{/Q}$. Moreover, $\Gamma(\pi_Q)$ is a bisimulation relation between S and $S_{/Q}$ if Q is a bisimulation relation between S and S.

Definition 2-1.6 (Alternating Simulation Relation [7]). *Let S_a and S_b be systems with $Y_a = Y_b$. A relation $R \subseteq X_a \times X_b$ is an alternating simulation relation from S_a to S_b if the following three conditions hold:*

1. *$\forall x_{a0} \in X_{a0} : \exists x_{b0} \in X_{b0}$ with $(x_{a0}, x_b 0) \in R$;*
2. *$\forall (x_a, x_b) \in R : H_a(x_a) = H_b(x_b)$;*
3. *$\forall (x_a, x_b) \in R$ and $\forall u_a \in U_a(x_a) : \exists u_b \in U_b(x_b)$ such that $\forall x'_b \in Post_{u_b}(x_b) : \exists x'_a \in Post_{u_a}(x_a))$ satisfying $(x'_a, x'_b) \in R$.*

If such a relation exists, it is said that S_a is alternatingly simulated by S_b, denoted by $S_a \preceq_{AS} S_b$.

Definition 2-1.7 (Extended Alternating Simulation Relation [7]). *Let R be an alternating simulation relation from system S_a to S_b. The extended alternating simulation relation $R^e \subseteq X_a \times X_b \times U_a \times U_b$ associated with R is defined by all the quadruples $(x_a, x_b, u_a, u_b) \in X_a \times X_b \times U_a \times U_b$ for which holds:*

1. *$(x_a, x_b) \in R$*

2. *$u_a \in U_a(x_a)$;*

3. *$u_b \in U_b(x_b)$ and $\forall x'_b \in Post_{u_b}(x_b) : \exists x'_a \in Post_{u_a}(x_a)$ satisfying $(x'_a, x_b \in R)$.*

Definition 2-1.8 (Feedback Composition [7])**.** *A system S_c is feedback composable with a system S_a if there exists an alternating simulation relation R from S_c to S_a. When S_c is feedback composable with S_a, the feedback composition of S_c and S_a, with interconnection relation $\mathcal{F} = R^e$, is given by $S_c \times_{\mathcal{F}} S_a$.*

2-1-1 Safety Games

Definition 2-1.9 (Safety Game [7])**.** *Let S_a be a system with $Y_a = X_a$ and $H_a = 1_x$, and let $W \subseteq X_a$ be a set of safe states. The safety game for system S_a and specification set W asks for the existence of a controller S_c such that:*

1. *S_c is feedback composable with S_a;*
2. *$S_c \times_{\mathcal{F}} S_a$ is nonblocking*
3. *$\emptyset \neq \mathcal{B}^\omega(S_c \times_{\mathcal{F}} S_a) \subseteq W^\omega$.*

Here $\mathcal{B}^\omega(S)$ denotes the infinite (external) behavior of S, which contains the infinite sequence of outputs that is allowed by the system S starting from any state x.

Consider the operator [7]:

$$F_W(Z) = \{x_a \in Z | x_a \in W \wedge \exists u_a \in U_a(x_a) \;\; \emptyset \neq Post_{u_a}(x_a) \subseteq Z\},$$

which contains all the states $x_a \in Z \cap W$ for which all the u_a-successors of x_a are in Z.

Proposition 2-1.1 ([7])**.** *The operator F_W satisfies:*

1. *$Z \subseteq Z' \implies F_W(Z) \subseteq F_W(Z')$;*
2. *If the safety game for system S_a and specification set W is solvable, then the maximal fixed-point Z of F_W satisfies $Z \cap X_{a0} \neq \emptyset$.*

A controller can be constructed as: $S_C = (X_c, X_{c0}, U_a, \xrightarrow[c]{})$, where:

- $X_c = Z$
- $X_{c0} = Z \cap X_{a0}$;
- $x_c \xrightarrow[c]{u_a} x'_c$ if $\emptyset \neq Post_{u_a}(x_c) \subseteq Z$.

Theorem 2-1.2 ([7])**.** *The maximal fixed-point Z of F_W can be obtained as:*

$$Z = \lim_{i \to \infty} F^i_W(X_a). \tag{2-6}$$

2-1-2 Partitioning and Refinement

The following definitions are from Section 7.3 of [8].

Definition 2-1.10 (Partition, Block, Superblock)**.** *A partition for set S is a set $\Pi = \{B_1, \dots, B_k\}$ such that $B_i \neq \emptyset, B_i \cap B_j = \emptyset \, (i \neq j)$ and $S = \bigcup_{0<i\leq k} B_i$.*
$B_i \in \Pi$ is called a block. $C \subseteq S$ is a superblock of Π if $C = B_{i_1} \cup \dots \cup B_{i_l}$ for some $B_{i_1}, \dots, B_{i_l} \in \Pi$.

Let $[s]_\Pi$ denote the unique block of partition Π containing s. A partition Π_1 is finer than Π_2, or Π_2 is coarser than Π_1 if: $\forall B_1 \in \Pi_1 \exists B_2 \in \Pi_2 : B_1 \subseteq B_2$.

Definition 2-1.11 (The Refinement Operator)**.** *Let Π be a partition for S and C be a superblock of Π. Then:*

$$Refine(\Pi, C) = \bigcup_{B \in \Pi} Refine(B, C),$$

where $Refine(B, C) = \{B \cap Pre(C), B \backslash Pre(C)\} \backslash \{\emptyset\}$

By repeatedly applying the refinement operator, the partition becomes increasingly finer, until it becomes a fixed-point for every superblock C. The induced equivalence relations R_{Π_i} satisfy:

$$S \times S \supseteq R_{\Pi_0} \supsetneqq R_{\Pi_1} \supsetneqq \dots \supsetneqq R_{\Pi_i} = R_{bisim.},$$

where $R_{bisim.}$ is a bisimulation relation between the maximally refined partitioned system and the original system. The initial partition Π_0 can be chosen as:

Definition 2-1.12 (The *AP* Partition)**.** *The AP partition of a transition system S denoted Π_{AP}, is the quotient space S/R_{AP} induced by $R_{AP} = \{(x_1, x_2) \in X \times X \mid H(x_1) = H(x_2)\}$.*

This is a rather natural choice since (alternatingly) (bi)similar states have the same output.

2-2 Event Triggered Control

Event Triggered Control (ETC) is an alternative for periodic control for implementation of control. Instead of recomputing the control action periodically, it is computed whenever a triggering condition is satisfied. An overview of different ETC techniques have been in [9], from which the linear ETC will be followed in this section. Starting with a linear plant

$$\dot{\xi}(t) = A\xi(t) + Bu(t), \tag{2-7}$$

where $\xi(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^{n_u}$, $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times n_u}$ and a linear static state-feedback controller

$$u(t) = K\xi(t), \tag{2-8}$$

where $K \in \mathbb{R}^{n_u \times n}$ such that the closed loop is asymptotically stable, the corresponding ETC system can be constructed as follows.
Suppose Lyapunov function $V(t) := \xi(t)^T P \xi(t)$ (with P symmetric positive definite) exists for the closed loop system 2-7, 2-8. This Lyapunov function has a time derivative of $\dot{V}(t) =$

$-\xi(t)^T Q\xi(t)$, where Q is guaranteed positive definite. A slower decay rate can be tolerated, which still assures asymptotic stability:

$$\dot{V}(t) \le -\sigma\xi(t)^T Q\xi(t), \tag{2-9}$$

where $\sigma \in (0,1]$. The idea is to recompute the control action only when 2-9 is about to be violated. Replace $u(t)$ by $\hat{u}(t) = u(t_k) = K\xi(t_k)$, where $\{t_k\}_{k\in\mathbb{N}}$ represent the times where the feedback law is recomputed, such that it effectively becomes a sample-and-hold controller. Then define the error

$$e(t) := \xi(t_k) - \xi(t),\ \forall t \in [t_k, t_{k+1}), k \in \mathbb{N}. \tag{2-10}$$

The time derivative of the Lyapunov function can then be written as:

$$\dot{V}(t) = -\xi(t)^T Q\xi(t) + 2\xi^T PBKe(t). \tag{2-11}$$

Combining this with the relaxed condition on the decay of the Lyapunov function results in:

$$\begin{bmatrix}\xi^T(t) & e^T(t)\end{bmatrix}\begin{bmatrix}(\sigma-1)Q & PBK \\ K^T B^T P & 0\end{bmatrix}\begin{bmatrix}\xi(t) \\ e(t)\end{bmatrix} =: z^T(t)\Psi z(t) =: \Gamma(z) \le 0. \tag{2-12}$$

This expression then becomes the triggering condition when equality holds. The matrix Ψ is called the triggering matrix and $\Gamma(\cdot)$ the triggering condition. Other variants for the triggering condition are possible, for example $\Gamma(z) = |e|^2 - \sigma|x|^2$. The trigger times generated by this condition are in general aperiodic and are defined by:

$$t_0 = 0,\ t_{k+1} = \inf\{t \in \mathbb{R} |\ t > t_k \wedge z(t)^T\Psi z(t) = 0\}. \tag{2-13}$$

Since the triggering conditions needs to be checking continuously, it is sometimes also called Continuous Event Triggered Control (CETC). Other implementations, such as Self-Triggered Control (STC), where the next triggering time depends on which region of the state space the system currently resides, output-based ETC and issues regarding for example Zeno behaviour can be found in [9].

2-2-1 Periodic Event Triggered Control

A similar class of systems, called Periodic Event Triggered Control, is constructed in a similar manner to regular ETC with one exception. Instead of continuously checking the triggering function, it is checked periodically. Here, the approach of [10] is followed. In PETC, the triggering times are integer multiples of the sampling period: $t_k = kh$, $k \in \mathbb{N}$ and h is the sampling period. The controller then takes the form $\hat{u}(t) = K\hat{\xi}(t),\ t \in \mathbb{R}_+$, where

$$\hat{\xi}(t_k) = \begin{cases}\xi(t_k), & \text{when } \Gamma(\xi(t_k), \hat{\xi}(t_k) > 0 \\ \hat{\xi}(t_k), & \text{when } \Gamma(\xi(t_k), \hat{\xi}(t_k) \le 0\end{cases} \tag{2-14}$$

Where $\Gamma(\cdot)$ is the triggering condition which is often times quadratic. The PETC system can then equivalently be expressed as:

$$\begin{aligned}\dot{\xi}(t) &= A\xi(t) + BK\hat{\xi}(t) \\ \xi(0) &= \hat{\xi}(0) = \xi_0.\end{aligned} \tag{2-15}$$

In general, the maximum inter-event time can be unbounded: $(t_{k+1} - t_k) \in h\mathbb{N}$. However, here we consider a maximum inter-event time hk_{max}, such that the next trigger time can be expressed as:

$$t_{i+1} = \inf\{t = kh > t_i,\, k \in \mathbb{N} \mid \Gamma(\xi(t), \hat{\xi}(t)) > 0 \vee t - t_i \geq hk_{max}\}. \tag{2-16}$$

A similar method using a Lyapunov function to before can be used to generate the triggering condition. However, this time it is required that the decay is low enough at least until the next sample, otherwise the system should trigger. It can be derived by first discretizing the dynamics 2-15:

$$\xi(t_{k+1}) = A_d\xi(t_k) + B_dK_d\hat{\xi}(t_k), \tag{2-17}$$

where

$$A_d := e^{Ah}, \text{ and } \quad B_d := \int_0^h e^{As}Bds, \tag{2-18}$$

and K_d is designed such that the closed loop discrete time dynamics are rendered stable. There exists a Lyapunov function $V(\xi(t_k)) = \xi(t_k)^TP\xi(t_k)$, where P is positive definite and

$$(A_d + B_dK_d)^TP(A_d + B_dK_d) \preceq \lambda P, \tag{2-19}$$

for some $0 \leq \lambda < 1$, implying that $V(\xi(t_{k+1})) \leq \lambda V(\xi(t_k))$. Requiring that the Lyapunov function decreases at least with some factor $0 \leq \beta < 1$, results a triggering condition with triggering matrix:

$$\Psi = \begin{bmatrix} A_d^TPA_d - \beta P & A_d^TPB_dK_d \\ K_d^TB_d^TPA_d & K_d^TB_d^TPB_dK_d \end{bmatrix}. \tag{2-20}$$

The advantage of this approach is that V is a Lyapunov function for the PETC system 2-15 with the triggering condition with matrix 2-20. This means that stability is inherently guaranteed.

2-2-2 Abstractions of PETC Systems

The triggering behaviour of a (Periodic) ETC system 2-15 can be abstracted by a transition system, which will be referred to as the traffic model of the (Periodic) ETC control loop. In this thesis, only abstractions for PETC will be considered, as the algorithms discussed will be directly applicable. Additionally, the considered abstractions will be limited to PETC system as discussed in Section 2-2-1, i.e. linear, undisturbed, driven by a static state-feedback controller and a quadratic triggering condition. Methods of constructing traffic models can be divided into two main categories:

- Space partitioning: Here the state space of the plant is first divided into regions (for example by isotropic partitioning [11]) and then computing the lower and upper bound for the triggering time for each of those regions. This is done for PETC in [12].
- Time partitioning: Here a (finite) list of triggering times $\{\tau_0, \ldots, \tau_n\}$ is specified, after which the corresponding regions in the state space are computed. For PETC this is done in [6]. This has the advantage over the previous method, because the number of regions does not grow exponentially with the dimension of the state space. Instead it grows with the number of triggering times considered.

In this thesis the time partitioning method is used, because of this scaling issue. In the rest of the section, this approach [6] is summarized.

The only three parameters that determine the resulting traffic model are the sampling period h, the maximum inter-event time k_{max} and the triggering matrix Ψ. Define

$$M(k) := e^{Akh} + \int_0^{kh} e^{A\tau} d\tau \, BK, \tag{2-21}$$

and

$$N(k) := \begin{bmatrix} M(k) \\ I \end{bmatrix}^T \Psi \begin{bmatrix} M(k) \\ I \end{bmatrix}. \tag{2-22}$$

Then the discrete inter-event time can be expressed as:

$$\kappa(x) = \min\{k \in \{1, 2, \ldots, k_{max}\} \,|\, x^T N(k) x > 0 \, \vee \, k = k_{max}\}. \tag{2-23}$$

The goal is to construct a quotient system model $\mathcal{S}_{/\mathcal{R}}$ of $\mathcal{S} = (X, X_0, \rightarrow, Y, H)$, where:

- $X = X_0 = \mathbb{R}^n$,
- $\rightarrow = \{(x, x') \in X \times X \,|\, x' = \xi_x(h\kappa(x))\}$,
- $Y = \{1, 2, \ldots, k\}$,
- $H = \kappa$,

which is basically an (uncountably) infinite-state transition system that captures the triggering behaviour. The state space can be partitioned by the sets:

$$\mathcal{K}_k := \begin{cases} \{x \in \mathbb{R}^n |\, x^T N(k) x > 0\}, & k < k_{max} \\ \mathbb{R}^n, & k = k_{max} \end{cases}, \tag{2-24}$$

describing the set of states that have at least triggered by time k. These sets can then be used to partition the state space as:

$$\mathcal{Q}_k = \begin{cases} \mathcal{K}_k \setminus \bigcup_{j=1}^{k-1} Q_j, & k > 1, \\ \mathcal{K}_k, & k = 1. \end{cases} \tag{2-25}$$

The state set of the quotient system is then $X_{/\mathcal{R}} = \{\mathcal{Q}_1, \mathcal{Q}_2 \ldots\}$. The size of this state set grows linearly with the number of allowed trigger times. The problem of whether there exists a transition between regions $\mathcal{Q}_i$ and $\mathcal{Q}_j$ is:

$$\exists x \in \mathbb{R}^n : x \in \mathcal{Q}_i, \, \xi_x(ih) = M(i)x \in \mathcal{Q}_j. \tag{2-26}$$

This can be expanded into a non-convex quadratic constraint problem

$$\begin{aligned} & \exists x \in \mathbb{R}^n \\ \text{s.t.} \quad & x^T N(i) x > 0, \\ & x^T N(i') x \leq 0, \, \forall i' \in \{1, \ldots, i-1\} \\ & x^T M(i)^T N(j) M(i) x > 0, \\ & x^T M(i)^T N(j') M(i) x \leq 0, \, \forall j' \in \{1, \ldots j-1\}. \end{aligned} \tag{2-27}$$

The non-convexity of this problem is a large disadvantage. However, it can be converted to a semi-definite problem. The transition set then consists of the pairs $(\mathcal{Q}_i, \mathcal{Q}_j) \in \mathcal{E}_{\setminus\mathcal{R}}$ which satisfy Equation (2-27).
For the scheduling of such systems, it is advantageous that a system can trigger earlier as well, i.e. for any state $\mathcal{Q}_i \in X_{\setminus\mathcal{R}}$ allow triggers at times $k \in \mathbb{N} : k < i$. The corresponding transitions can then be computed by verifying Equation (2-27), replacing j by k, resulting in the transitions $(\mathcal{Q}_i, k, \mathcal{Q}_j) \in \mathcal{E}^*$. These will be referred to as early triggers. The previously defined transitions are also modified to make the trigger time explicit: $(\mathcal{Q}_i, i, \mathcal{Q}_j) \in \mathcal{E}_{\setminus\mathcal{R}}$

This results in a traffic model $S_{\setminus\mathcal{R}} = (X_{\setminus\mathcal{R}}, X_{\setminus\mathcal{R},0}, U_{\setminus\mathcal{R}}, \mathcal{E}_{\setminus\mathcal{R}} \cup \mathcal{E}^*, Y_{\setminus\mathcal{R}}, H_{\setminus\mathcal{R}})$, where:

- $X_{\setminus\mathcal{R}} = X_{\setminus\mathcal{R},0} = \{\mathcal{Q}_1, \dots, \mathcal{Q}_{k_{max}}\}$,
- $\mathcal{E}_{\setminus\mathcal{R}}$ and $\mathcal{E}^*$ as defined above,
- $U_{\setminus\mathcal{R}} = Y_{\setminus\mathcal{R}} = \{1, \dots, k_{max}\}$,
- $H_{\setminus\mathcal{R}}(\mathcal{Q}_k) = k$.

In addition to early triggers, also transitions for late triggers can be calculated, i.e. transitions $(\mathcal{Q}_i, k, \mathcal{Q}_j) \in \mathcal{E}_{late}$, $k \in \mathbb{N} : k > i$ that satisfy Equation 2-27. These transitions are not used for the main algorithms that will be discussed, since they do not guarantee stability inherently. However, the transitions $\mathcal{E}_{late}$ will be added to the traffic model in the approach in Chapter 5.

2-3 Binary Decision Diagrams

Binary Decision Diagrams (BDDs) [13, 14] are directed acyclic graphs used to represent boolean functions $f \colon \mathbb{B}^n \to \mathbb{B}$. Each node v in a BDD corresponds to a boolean variable $var(v) = x_i$ and has two children: $low(v)$ corresponding to $x_i = 0$ and $high(v)$ corresponding to $x_i = 1$. A terminal node is either labelled 0 or 1. To construct the BDD, Shannon's expansion

$$f = x_i \cdot f|_{x_i=1} + \bar{x_i} \cdot f|_{x_i=0}$$

can be used to iteratively generate the BDD starting from the first boolean variable x_1. An example of a BDD is shown in Figure 2-1.

In addition to Boolean functions, BDDs can also be used to represent sets and functions on finite domains [15]. Assume set A is encoded into $n = \lceil log_2 |A| \rceil$ binary variables using the encoding function $\sigma \colon A \to \{0,1\}^n$, with $\sigma_i(\cdot)$ denoting the ith bit of the encoding. A function $f \colon A \to A$ can be represented with n Boolean functions $f_i \colon \{0,1\}^n \to \{0,1\}$ defined as $f_i(\sigma(a)) = \sigma_i(f(a))$. A set $S \subseteq A$ can then be represented using the Boolean function $\chi_S \colon \{0,1\}^n \to \{0,1\}$, defined as:

$$\chi_S(x) = \sum_{a \in S} \prod_{1 \leq i \leq n} [x_i \iff \sigma_i(a)], \tag{2-28}$$

The function χ_S is also called the characteristic function for the set S.

2-3-1 Ordered Binary Decision Diagrams

The ordering of variables and nodes in a BDD can have significant impact on its size. Because of this Ordered Binary Decision Diagrams (OBDDs) have been introduced [16, 15], in which a certain ordering over the boolean variable is enforced: Given an ordering $<$ over the boolean variables, then for any vertex v and either nonterminal child v, $var(u) < var(v)$ must hold. The size of an OBDD can potentially be reduced further by three transformation that do not change the function represented:

- Remove duplicate terminal nodes and redirect all the arcs into the removed nodes to the remaining one.
- If nodes u and v have $var(u) = var(v)$, $low(u) = low(v)$ and $high(u) = high(v)$, then remove one of them and redirect all its incoming arcs to the other node.
- If node v has $low(v) = high(v)$ then remove v and redirect all incoming arcs to $low(v)$.

When these transformations are repeated until the resulting OBDD no longer changes, it is called a Reduced Ordered Binary Decision Diagram (ROBDD). In the rest of this thesis when referring to a BDD it is implied that it is an also a ROBDD.

2-3-2 Operations on (RO)BDDs

An advantage of BDDs is that operations on the original Boolean function can analogously be directly applied to the BDD. Additionally, the following operations [16, 15] preserve the ordering of the original OBDD. However, in general the resulting OBDD is not a ROBDD and it has to be reduced again.

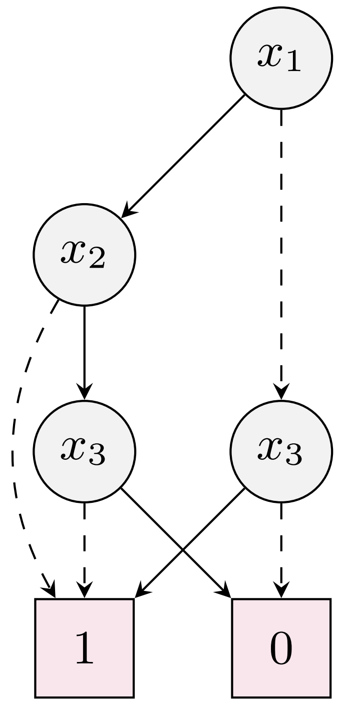


Figure 2-1: An example BDD representing the boolean function $f = x_1\bar{x}_2+\bar{x}_1x_3+\bar{x}_3x_1x_2$. Each node is labelled with its corresponding boolean variable. The solid lines represent that the variable is 1, the dashed lines represent a 0. The value of f can be calculated by going through the diagram and taking the edge corresponding to the value of the boolean variable. The value of f is then the value of the terminal node that is reached.

APPLY

Given a binary Boolean operator $\star$ and two Boolean functions f and g, the APPLY operation generates a new Boolean function: $h = f \star g$. This can be expanded using the Shannon expansion for any variable x:

$$f \star g = \bar{x} \cdot (f|_{x \leftarrow 0} \star g|_{x \leftarrow 0}) + x \cdot (f|_{x \leftarrow 1} \star g|_{x \leftarrow 1}). \tag{2-29}$$

This can be used to recursively generate the OBDD representation of $f \star g$. For more details and a more efficient implementation see [16, 15].

Restriction

The restriction operation transforms the BDD representing f into the BDD representing $f|_{x_i=b}$ by traversing the original BDD and looking for all arcs towards x_i and changing those to either $low(x_i)$ when $x_i = 0$ or $high(x_i)$ when $x_i = 1$. Finally the BDD is reduced back into a ROBDD.

Composition

Composition of two functions f_1 and f_2, represented by BDDs, can be described using the Shannon expansion:

$$f_1|_{x_i=f_2} = f_2 \cdot f_1|_{x_i=1} + (\neg f_2) \cdot f_1|_{x_i=0} = ITE(f_2, f_1|_{x_i=1}, f_1|_{x_i=0}),$$

where ITE(if-then-else) is a ternary Boolean operation defined by:

$$ITE(a, b, c) = a \cdot b + (\neg a) \cdot c,$$

which can be applied using an extension of the APPLY operation. See [15] for more details.

Remarks

Often the operators $\exists x$ and $\forall x$ are also used, these can be expanded into:

$$\begin{aligned} \exists x.f &= f|_{x \leftarrow 0} \vee f|_{x \leftarrow 1} \\ \forall x.f &= f|_{x \leftarrow 0} \wedge f|_{x \leftarrow 1} \end{aligned} \tag{2-30}$$

The APPLY operation can then be used to resolve these expressions In the case that $x = (x_0, x_1, \dots)$, all the different combinations are combined depending on the operator. For example if $x = (x_0, x_1)$, then

$$\exists x.f = f|_{x \leftarrow (0,0)} \vee f|_{x \leftarrow (0,1)} \vee f|_{x \leftarrow (1,0)} \vee f|_{x \leftarrow (1,1)}. \tag{2-31}$$

Most of the time in this thesis, composition is used as a tool to require that two BDDs both evaluate to true. This is accomplished by replacing the terminal node 1 of f_1 by f_2. This is then equivalent to composition with $node(x_i) = 1$, such that the composition just becomes $f_1|_{x_i=f_2} = f_1 \cdot f_2$.

Chapter 3

Scheduling of Periodic Event Triggered Control Systems

In this chapter two algorithms for synthesizing schedulers for systems of Periodic Event Triggered Control (PETC) subsystems are discussed. These schedulers have two requirements:

- The scheduler needs to render each subsystem stable, by triggering each subsystem in time.
- The scheduler has to prevent a 'collision' of triggering events, i.e. not more than one subsystem can trigger simultaneously.

First it is discussed how to construct the PETC systems for which the scheduler will be synthesized. Secondly, it is described how the synthesization is performed by means of a safety game. This will be the first algorithm. Thirdly, a way to reduce the size of the subsystems is discussed, as well as the second algorithm which makes use of this. The procedures using Binary Decision Diagrams (BDDs) are discussed in Chapter 4.

3-1 Construction of Abstractions

Given a set of m PETC systems, described by:

$$\begin{aligned} \dot{\xi}_i(t) &= A_i\xi_i(t) + B_iK_i\hat{\xi}_i(t), \quad \text{and} \\ \Gamma_i(z_i(t)) &= \begin{bmatrix}\xi_i^T(t) & \hat{\xi}_i^T(t)\end{bmatrix} \Psi_i \begin{bmatrix}\xi_i(t) \\ \hat{\xi}_i(t)\end{bmatrix}, \quad i \in \{0,\dots,m-1\}. \end{aligned} \tag{3-1}$$

The goal is to construct a model which abstracts and incorporates all the control loops. Traffic models $S_{\backslash\mathcal{R},i}$ for each of these control loops are constructed as is discussed in Section 2-2-2. Other abstraction methods can be used as well to generate these traffic models (not restricted to PETC), however the resulting models should satisfy some conditions:

- The traffic model should be finite-dimensional, and the actions must represent the next triggering time, which additionally must be a multiple of the sample time h. This is automatically satisfied when considering PETC.
- Here it is assumed that no two states in the traffic model share the same maximum trigger time. While this will not hold for general traffic models, it is the case for the abstraction algorithm described in Section 2-2-2, as the state space is split based on the maximum trigger time.
- All the subsystems must share the same sampling times: $h_0 = \cdots = h_{n-1} = h$, because then all the actions take place at the same time.

Each of the traffic models is first converted into a Transition System (TS) (with a certain structure), and subsequently they should be composed into a larger system. This is discussed in the next two subsections.

3-1-1 Converting Traffic Models

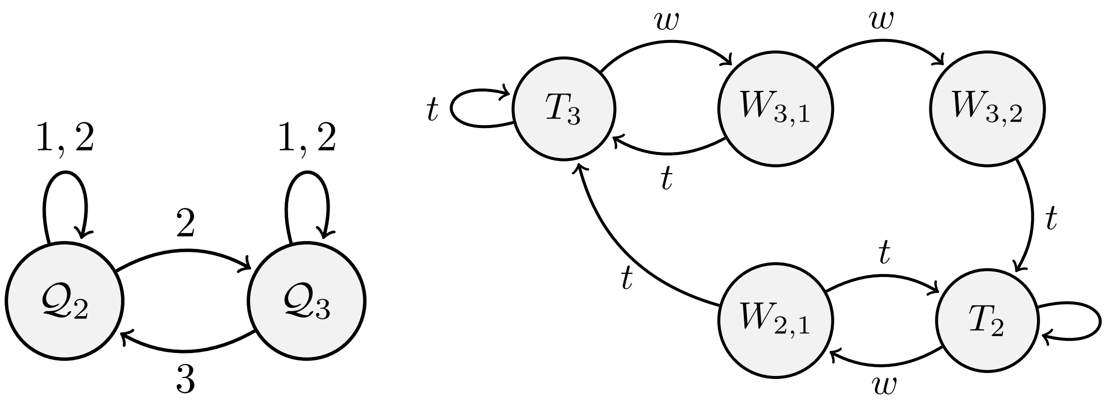
Before a scheduler can be synthesized on the traffic models $S_{\backslash\mathcal{R},i}$, they are first converted to a different representation. A transition $(\mathcal{Q}_i, k, \mathcal{Q}_j)$ in the traffic model can be represented by a sequence of actions: $\mathcal{B}_k = w, w, \ldots, t$, where the number of w actions is $k-1$, such that the system triggers after k samples. This representation is possible because the sampling times are periodic with period h resulting in inter-event times that are integer multiples of the sampling period: $\tau_k = t_{i+1} - t_i = kh$ for some $k \in \mathbb{N}$, $0 < k \leq k_{max}$. All possible sequences $\mathcal{B}_k$ can be generated by a TS S that is constructed as follows:

1. For every state $\mathcal{Q}_i$ in $S_{\backslash\mathcal{R}}$ do:
   (a) Create state T_i in the S, with output $H(T_i) = T$ if $i > 1$ else $H(T_i) = T_1$.
   (b) If $i > 1$: create states $W_{i,j}$, $1 \leq j < i$, with outputs $H(W_{i,j}) = W_{i-j}$.
2. For every transition $(\mathcal{Q}_i, k, \mathcal{Q}_j)$ in $S_{\backslash\mathcal{R}}$ do:
   (a) If $k > i$: skip the transition, else
   (b) If $k = 1$: Create transition (T_i, t, T_j) in S.
   (c) Else: Create transitions $(T_i, w, W_{i,1}), \ldots (W_{i,k-1}, t, T_t)$ in S.

The outputs and output map is defined in such a way that partitioning according to the outputs results in a useful partition. More on this in Section 3-3-1. An example of this process is shown in Figure 3-1. If not explicitly specified or if it is clear from context, when using the term subsystem or TS, usually the representation above is implied.

3-1-2 Composition

Before synthesization can start, all subsystems to be considered must be composed into a single system. This can be done by defining interconnection relation $\mathcal{I}_{par.} := X_n \times \cdots \times$



(a) An example traffic model. **(b)** The equivalent TS.

Figure 3-1: An example of converting a traffic model to a TS.

$X_1 \times U_n \times \cdots \times U_1$, which will be referred to a parallel interconnection relation. Using Definition 2-1.2 of composition, a composed system can be constructed. Intuitively, this allows all combinations of transitions of the subsystems to occur without interactions between them, effectively letting every subsystem run in parallel. The composed system is then $S := S_n \times_{\mathcal{I}_{par.}} \cdots \times_{\mathcal{I}_{par.}} S_0$, where the descriptor $\times_{\mathcal{I}_{par.}}$ is often dropped and all the compositions are contracted as $S := \bigtimes_i S_i$. The space complexity of this composed system is $\sim O(\prod_i N_T^i)$, where N_T^i is the number of transitions in subsystem i. As can be seen the composed system will grow very quickly with the number of subsystems. An example composition is shown in Figure 3-2.

3-2 Standard Scheduler Synthesization

A scheduler is synthesized on the composition of subsystems by solving a safety game. This is done by iterating over the operator [7]:

$$F_W(Z) = \{x \in Z \,|\, x \in W \wedge \exists u \in U(x) : \emptyset \neq Post_u(x) \subseteq Z\}, \tag{3-2}$$

where W is the safe set for which the game is solved. In this case, at most one state can be in a trigger state or equivalently in a state which has output $H(x_i) \in \{T_1, T\}$. The safe set can then be expressed as:

$$W = \{(x_1, \ldots, x_n) \in X \,|\, \exists_{\leq 1} x_i : H_i(x_i) \in \{T_1, T\}\}. \tag{3-3}$$

The safety game is then solved by initializing $Z := X$ and iterating $Z \leftarrow F_W(Z)$ until it does not change: $Z^* = F_W(Z^*)$. A solution has been found if $Z \neq \emptyset$, otherwise the scheduling problem is not feasible. Using this solution, a controller system which is feedback composable can be constructed [7]: $S_c = (X_c, X_{c0}, U_c, \underset{c}{\rightarrow})$, where:

- $X_c = Z$;

- $X_{c0} = Z \cap X_0$
- $x_c \underset{c}{\xrightarrow{u}} x'_c \quad \text{if} \quad \emptyset \neq Post_u(x_c) \subseteq Z$

Here $Post_u(x)$ refers to the post set of system S for input u and state x, which is valid because the controller state set is a subset of the system state set. The controller state set can be modified to include all states (even unsafe states): $X_c = X$. The advantage of this is if the system starts in an unsafe state or some other state outside Z, the scheduler can still find an input if it leads to a state in Z.
However, only the possible safe inputs are needed to safely schedule the subsystems and the scheduler can be a somewhat simplified version of the controller system. The scheduler only needs to choose an input from $U_c(x)$ which is defined as:

$$U_c(x) = \{u \in U \mid \exists x' : (x, u, x') \in \underset{c}{\rightarrow}\} \tag{3-4}$$

An example of the safety game is shown in Figure 3-3 for a very simple system, taking only two iterations until a solution has been found.

3-3 Alternative Synthesization Algorithm by Partitioning

The generated TSs contain a lot of structure, in which a lot of states have similar behaviour. It makes sense to group these states together into blocks and creating a new TS from these, with the advantage of greatly reducing the number of states to be considered. This would improves the efficiency of the safety game. However, it is not guaranteed a scheduler can be found for these partitioned systems, even if the safety game is solvable for the original system. To resolve this issue, partitioned subsystems can be refined by splitting apart blocks, allowing more behaviour and hopefully solving the safety game. This section describes how the subsystems are partitioned, refined and how the synthesization algorithm is modified to include partitioning and refinement.

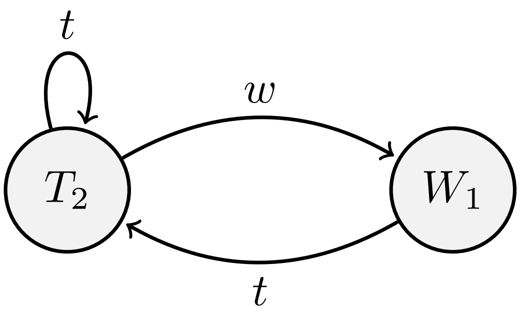
3-3-1 Initial Partition

Partitioning of the TS follows the definition of the AP partition (Definition 2-1.12). Each block b is associated to an output o of the TS. Then a state $s \in \bar{X}$ lies in b if $H(s) = o$. These blocks form the set of states for the partitioned TS S^I: $X^I = \{b_0, b_1, \dots\}$. The transitions in S^I are:

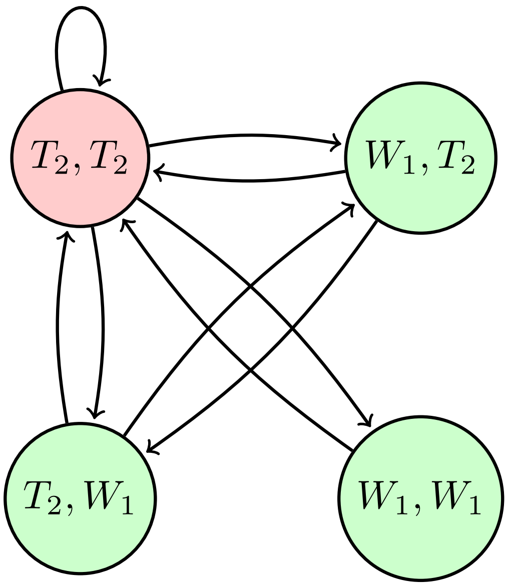
$$\underset{S^I}{\longrightarrow} := \{ (b, u, c) \mid \exists (x, u, y) \in \underset{\bar{S}}{\longrightarrow} : x \in b \wedge y \in c\}, \tag{3-5}$$

The output of block b associated with output o is o: $H(b) = o$. This defines a new TS: $S^I = (X^I, X^I_0, U^I, \underset{S^I}{\longrightarrow}, O^I, H^I)$, where

- $X^I = \{b_1, b_2, \dots\}$,
- $X^I_0 = \{b \in X^I \mid \exists x \in \bar{X} : x \in b\}$,

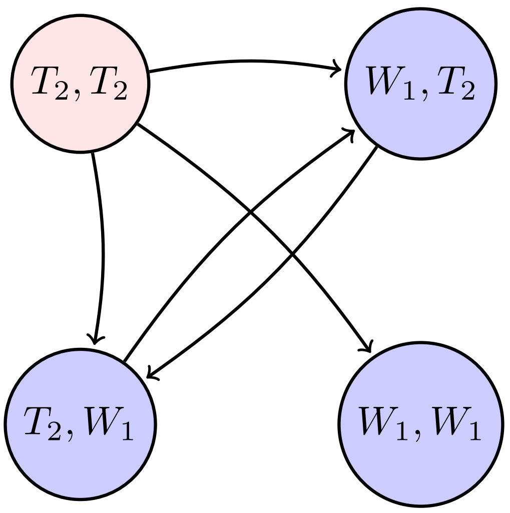


(a) The system that is used for the composition example.

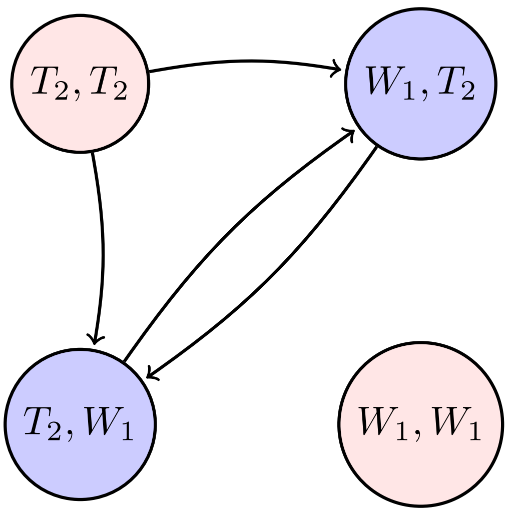


(b) The composition of two identical copies of 3-2a. The transitions are not labelled for legibility. The states which lie in the safe set are coloured green and the ones not in the safe set are coloured red.

Figure 3-2: An example of composing two identical systems.



(a) Result after one iteration: $Z_1 := F_W(Z_0) = W$.



(b) Result after two iterations: $Z_2 := F_W(Z_1) = \{(T_2, W_1), (W_1, T_2)\}$. More iterations do not have any effect so this is the solution to the safety game.

Figure 3-3: An example of solving the safety game on the composed system in Figure 3-2b. The states that lie in $Z^+ := F_W(Z)$ are coloured blue, the others are coloured red. The safety game is initialised with $Z_0 = X^2$. The transitions leading to any state in Z are kept while the others are removed.

- $U^I = \bar{U}$,
- $O^I = \bar{O}$,
- $\xrightarrow[S^I]{}$ as above, and
- $H^I(b) = \bar{H}(s)$, for any $s \in b$.

The partitioned system S^I is equivalent to the quotient system of $\bar{S}$ with equivalence relation

$$Q := \{(x, y) \in \bar{X} \times \bar{X} \mid \bar{H}(x) = \bar{H}(y)\} \tag{3-6}$$

Thus the state set can be alternatively expressed as $X^I := \{[x]_Q \,|x \in \bar{X}\}$. An example of partitioning a TS is shown in Figure 3-4.

In general, partitioning greatly reduces the size of the resulting TS: The number of states are reduced from $\left|\bar{X}\right| = O(k_{max}^2)$ to $\left|X^I\right| = \left|\bar{O}\right| = O(k_{max})$. The number of transitions is reduced from $\left|\xrightarrow[\bar{S}]{}\right| = O(\left|\bar{X}\right|^2 \cdot \left|\bar{U}\right|) = O(k_{max}^4)$ to $\left|\xrightarrow[S^I]{}\right| = O(\left|X^I\right|^2 \cdot \left|\bar{U}\right|) = O(k_{max}^2)$. This size reduction is advantageous for the size of the complete system, and thus will result in a faster synthesization time.

However, it is not immediately clear that schedulers synthesized on these partitioned systems are usable for the original systems. To show this, first consider the following proposition:

Proposition 3-3.1. *A system S, as constructed in Section 3-1-1, is alternatingly simulates its partitioned version S^I (its construction as described above), i.e.:*

$$S^I \preceq_{AS} \bar{S}. \tag{3-7}$$

The alternating simulation relation is:

$$R = \{(b, s) \in X^I \times \bar{X} \mid s \in b\} \tag{3-8}$$

A proof of Proposition 3-3.1 is given in Appendix B-1.

Additionally consider the following proposition:

Proposition 3-3.2 (From Section 8.2 of [7])**.** *Let S_a, S_b, and S_c be systems with the same output set, assume that S_c is feedback composable with S_a, and let ${}_cR_a$ be the corresponding alternating simulation relation. If there exists an alternating simulation relation ${}_aR_b$ from S_a to S_b, then $S_c \times_{{}_cR_a^e} S_a$ is feedback composable with S_b and the corresponding alternating simulation relation is given by:*

$${}_{ca}R_b = \{((x_c, x_a), x_b) \in (X_c \times X_a) \times X_b |\, (x_c, x_a) \in {}_cR_a \wedge (x_a, x_b) \in {}_aR_b\}. \tag{3-9}$$

This can be summarized by the following:

$$S_c \preceq_{AS} S_a \wedge S_a \preceq_{AS} S_b \implies S_c \times_{\mathcal{F}} S_a \preceq_{AS} S_b. \tag{3-10}$$

Propositions 3-3.1 and 3-3.2 together prove that synthesizing a scheduler for the partitioned system results in a scheduler S_c^I that can be used also on the original system: $\bar{S}_c := S_c^I \times_{\mathcal{F}} S^I \preceq_{AS} \bar{S}$. Here $\mathcal{F} = R^e$, where R^e is the extended alternating simulation relation based on the alternating simulation relation (3-8). In general this will result in a very unwieldy scheduler, however, similar to before, knowing the possible safe inputs is sufficient:

$$\bar{U}_c(x) = U_c^I(b), \text{ if} x \in b. \tag{3-11}$$

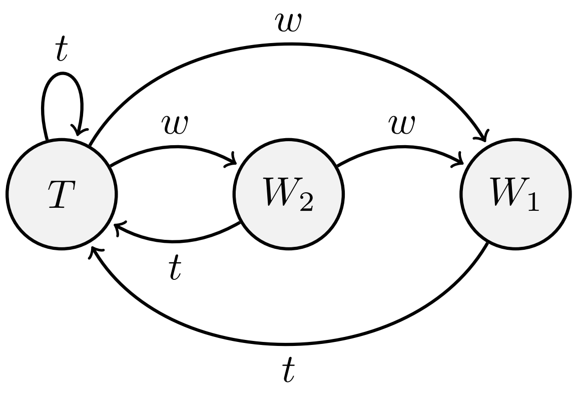


Figure 3-4: An example partitioning generated from the subsystem in Figure 3-1b.

3-3-2 Refinement

Synthesizing a scheduler on a collection of partitioned subsystems may not always be feasible, even though a scheduler does exist for the original non-partitioned subsystems. This can be explained intuitively by the fact that in some sense information is lost when partitioning. To resolve this issue, one can refine the partitioned TS a little to allow for more behaviours, which in turn makes the collection of subsystem more schedulable.

Use $S^{(i)}$ to denote system S after partitioning and $i-1$ refinements and use $\bar{S}$ to denote the original system. Similarly for all other related elements, e.g. the state set $X^{(i)}$. The refinement process uses the refinement operator 2-1.11:

$$Refine(X^{(i)}, C) := \bigcup_{B \in X^{(i)}} Refine(B, C), \tag{3-12}$$

where $C \subset X^{(i)}$ and

$$Refine(B, C) = \{B \cap Pre(C),\, B \setminus Pre(C)\} \setminus \{\emptyset\}. \tag{3-13}$$

Effectively this splits each block of the partitioned system into a part that does have transitions to C and a part that does not have transitions to C. This refinement operator is applied repeatedly for every state in the old system $S^{(i)}$, finally resulting in a new state set $X^{(i+1)}$. To make this into a TS, new transitions have to be computed as follows:

$$(x^{(i+1)}, u, y^{(i+1)}) \in \xrightarrow[S^{(i+1)}]{} \quad \text{if} \quad \exists (\bar{x}, u, \bar{y}) \in \xrightarrow[\bar{S}]{} : \quad \bar{x} \in x^{(i+1)} \wedge \bar{y} \in y^{(i+1)},$$

and the new initial state set and output map have to be computed by 'projection':

$$\begin{aligned} X_0^{(i+1)} &:= \{x \in X^{(i+1)} \mid \exists x_0 \in \bar{X}_0 : \, x_0 \in x\} \\ H^{(i+1)}(x) &:= \bar{H}(y) \quad \text{if} \quad y \in x \end{aligned} \tag{3-14}$$

This process has been summarized in Algorithm 3.1. An example of refining a partitioned TS is shown in Figure 3-5.

As before with partitioning, controllers designed on refined systems can be used on the original system. To show this first consider the following proposition:

Algorithm 3.1: Refinement of a system $S^{(i)}$.

```
Input: $S^{(i)}$: Current partitioned system, $\bar{S}$: The original system.
Output: $S^{(i+1)}$: Refined system
$X^{(i+1)} \leftarrow X^{(i)}$;
for $C \in X^{(i)}$ do
    $X^{(i+1)} \longleftarrow Refine(X^{(i+1)}, C)$;
$T \leftarrow \{(x^{(i+1)}, u, y^{(i+1)}) | (\bar{x}, u, \bar{y}) \in \underset{\bar{S}}{\longrightarrow} \wedge \bar{x} \in x^{(i+1)} \wedge \bar{y} \in y^{(i+1)}\}$;
$X_0^{(i+1)} \leftarrow \{x \in X^{(i+1)} | \exists x_0 \in \bar{X}_0 : x_0 \in x\}$;
$H^{(i+1)}(x) \leftarrow \bar{H}(y)$ if $y \in x$;
return $S^{(i+1)} := (X^{(i+1)}, X_0^{(i+1)}, U, T, Y, H^{(i+1)})$
```

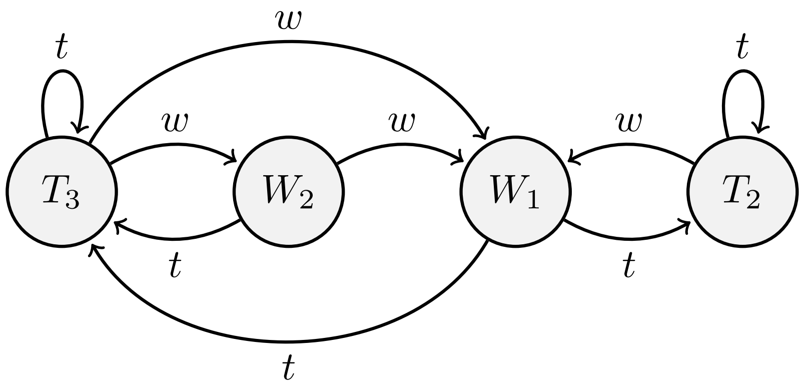


Figure 3-5: An example refinement generated from the subsystem in Figure 3-4 using block W_1. Two new blocks are created: $T_3 = \{T_3\}$ and $T_2 = \{T_2\}$. The blocks W_2 and W_1 remain the same.

Proposition 3-3.3. *Consider a system S that is partitioned once and refined $i-1$ times resulting in $S^{(i)}$, and the system $S^{(i+1)}$ that is the refined version of $S^{(i)}$. Then $S^{(i+1)}$ is alternatingly simulation by $S^{(i)}$, i.e.:*

$$S^{(i+1)} \preceq_{AS} S^{(i)}. \tag{3-15}$$

A proof for Proposition 3-3.3 is given in Appendix B-2. Then by transitivity of $\preceq_{AS}$ and Proposition 3-3.1 (recall that $S^{(0)} := S^I$), it is proved that $S^{(i+1)} \preceq_{AS} S$. Then, again by Proposition 3-3.2, a controller designed on the refined system can be used for the original system. Similarly to before only knowing the safe inputs is sufficient:

$$\bar{U}_c(x) = U_c^{(i)}(b^{(i)}), \text{ if } x \in b^{(i)}. \tag{3-16}$$

Remarks Although alternating simulation holds for partitioning and refining for individual subsystems, it must also hold for composition to allow schedulers synthesized on a composition of partitioned subsystem to be used on the original system. However, by the following proposition, schedulers synthesized on composed system where some of the subsystems are partitioned can be used on the original composed system as well:

Proposition 3-3.4. *[From Section 4.4 of [7]] Alternating simulation commutes with composition: If $S_a \preceq_{AS} S_b$ and $S_c \preceq_{AS} S_d$, then $S_a \times_{\mathcal{I}} S_c \preceq_{AS} S_b \times_{\mathcal{J}} S_d$, $\mathcal{J}$ can be determined from $\mathcal{I}$ and the alternating simulation relations from S_a to S_b and from S_c to S_d.*

This proposition can be used recursively to show that a composed system where a subset of the subsystems is partitioned is alternatingly similar to the original composed system (also by noting the fact that $S \preceq_{AS} S$).

Algorithm 3.2: Alternative Synthesization Algorithm

```
Input: $TSA = \{s_1, s_2, \dots\}$: List of PETC subsystems represented as TSAs.
Output: Scheduler for the composed system, represented by $U_c(x)$
$T \longleftarrow \emptyset$;
for $i \in TSA$ do
    $p \longleftarrow ToNFA(i)$;
    $T \longleftarrow T \cup InitPartition(p)$;
while not success do
    $sys \longleftarrow Compose(T)$;
    $W \longleftarrow SafeSet(sys)$;
    $U_c(x) \longleftarrow SafetyController(sys, W)$;
    if $\forall x \;\; U_c(x) = \emptyset$ then
        for $t \in T$ do
            $t \longleftarrow Refine(t)$
        if Refinement not successful then
            return $\emptyset$
    else
        return $U_c(x)$
```

3-3-3 Modified Scheduler Synthesization Algorithm

The synthesization algorithm will be modified as follows: First all the subsystems are partitioned. Subsequently they are composed and a scheduler is synthesized on the composition. If it is successful, a controller (which is expressed in terms of the blocks) is returned. Otherwise all systems are refined and synthesization is tried again. This continues until either a scheduler is found or further refinement is not possible. This procedure is summarized in Algorithm 3.2, and is applicable with or without BDDs. The advantage of this algorithm is that the size of the subsystems decrease significantly after partitioning while keeping a lot of the behaviour. This allows for much faster composition as well as a faster solution of the safety game (requiring both less iterations as well as faster iterations). However, if synthesization fails on the partitioned subsystems, refinement has to occur which increases the computational cost considerably. It is possible that a lot of refinement-synthesization loops have to occur before synthesization is successful causing the original algorithm to be faster (but requiring more memory). However, if a scheduler exists, it will also be found using this modified algorithm. To show this, the following proposition is used:

Proposition 3-3.5. *Given a system S for which the safety game (Definition 2-1.9) has a solution for some safe set W, and another system S'. Suppose $S' \cong S$ holds, then there also exists a solution to the safety game for S' a suitable safe set W'.*

A proof for Proposition 3-3.5 is given in Appendix B-3. After maximally refining a system S, resulting in system $S^{(\infty)}$, it will become bisimilar to S: $S^{(\infty)} \cong S$ (See Section 2-1-2). Thus if the safety game is feasible for the original system, it will also be found using Algorithm 3.2.

Finally, the resulting scheduler is expressed in terms of the blocks of the partition, meaning it has to be either converted back into a representation of states or a lookup table has to be provided to convert the states into blocks.

Remarks The alternative synthesization algorithm as prescribed above partitions and refines all systems simultaneously. However, it is also possible to partly partition or refine some subsystems, while keeping other in their original form. It is possible to conceive algorithms that perform refinement on subsystems based on some metric to slightly increase computational efficiency and possibly scheduler performance. More on this in the discussion (Section 8).

Chapter 4

Scheduler Synthesis Using Binary Decision Diagrams

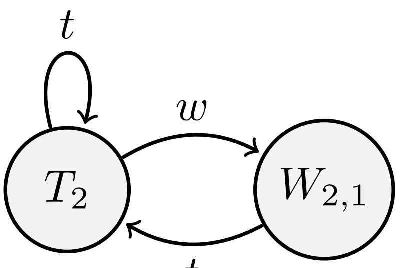
This chapter will discuss how to make the synthesization procedure more efficient by means of Binary Decision Diagrams (BDDs). First will be discussed how to encode the transition systems acquired from the PETC abstractions into BDDs. Then all the operations that make up both synthesization algorithm are expressed using boolean functions and BDDs such that a scheduler can be synthesized using BDD. Lastly, a rough comparison is made between the multiple methods by means of a time complexity analysis.

4-1 Representation by BDDs

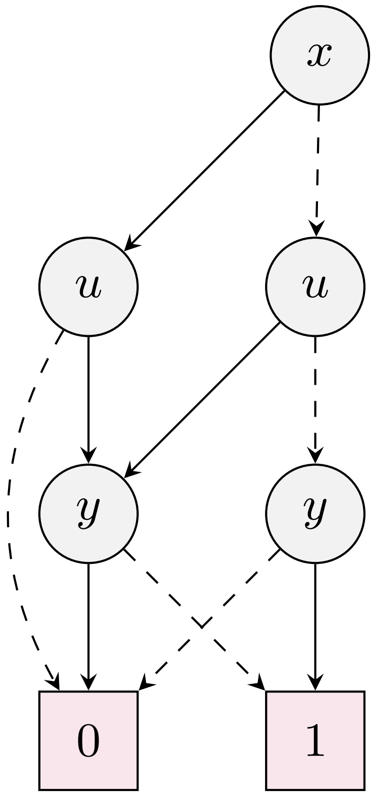
A transition systems can be represented by a BDD by encoding its states and actions by binary variables $x_{BDD} := (x_{n-1}, \ldots, x_0) \in \mathbb{B}^{n_x}$, where $n_x := \lceil \log_2 |X| \rceil$ and $u_{BDD} := (u_{n-1}, \ldots, u_0) \in \mathbb{B}^{n_u}$, where $n_u := \lceil \log_2 |U| \rceil$ and representing the transition set by a boolean function which in turn can be represented by a BDD. The process of representing the transition set by a boolean function is as follows:

1. Define n_x and n_u as above,
2. Define encoding function $\sigma_i^x(x)$ for the states and encoding function $\sigma_i^u(u)$ for the inputs.
3. Define three sets of boolean variables: $x := (x_{n-1}, \ldots, x_0) \in \mathbb{B}^{n_x}$, $u := (u_{n-1}, \ldots, u_0) \in \mathbb{B}^{n_u}$, $y := (y_{n-1}, \ldots, y_0) \in \mathbb{B}^{n_x}$.
4. Finally, the boolean function representing is a modification of Equation 2-28 (taking into account multiple variables) and is written as:

$$\chi_T(x, u.y) = \sum_{(i,u,t)\in\rightarrow} \left\{ \prod_{k=0}^{n_x-1} [x_k \Leftrightarrow \sigma_k^x(i)] \right\} \wedge \left\{ \prod_{k=0}^{n_u-1} [u_k \Leftrightarrow \sigma_k^u(u)] \right\} \wedge \left\{ \prod_{k=0}^{n_x-1} [y_k \Leftrightarrow \sigma_k^x(t)] \right\}, \tag{4-1}$$



(a) A very simple TS used as reference for representation by a BDD in Figure 4-1b.



(b) The BDD representing the TS in Figure 4-1a. The states and inputs are encoded as follows: $\sigma^x(T_2) = 0$, $\sigma^x(W_{2,1}) = 1$ and $\sigma^u(w) = 0$, $\sigma^u(t) = 1$.

Figure 4-1: An example of representing a TS by a BDD.

Even though only χ_T is necessary, for convenience a second boolean function $\chi_{tr}(x)$ is constructed. This boolean function describes whether the state x is a transmit state, i.e. whether $H(x) \in \{T_1, T\}$. It can be constructed by using Equation 2-28 and defining set $\mathcal{A}$ as:

$$\mathcal{A} := \{x \in X | \, H(x) \in \{T_1, T\}\}.$$

Representing the transitions set by boolean functions (and those by BDDs) has the advantage of decreasing the effective states and transitions or variables that have to be manipulated. This results in a lower time and space complexity. Without using BDDs, the space complexity of a Transition System (TS) is $O(|X|^2 \cdot |U|) \approx O(N_T)$. Using BDDs, the space complexity becomes $O(2n_x + n_x) = O(2\log_2|X| + \log_2|U|) \approx O(\log_2 N_T)$. Thus using BDDs will require much less memory for large systems, however, new points of overhead will also appear. These new overheads do not have significant impact on the synthesization process. However, the most impact will occur in the online setting, where actions satisfying a given BDD will have to be generated. More on this in Section 6-2.

Time complexity will be discussed in Section 4-6 as this depends on multiple systems as well as the algorithms used.

For the rest of this thesis, both the elements of the state set and their binary encodings will be represented by the same variable x for notational convenience. Likewise for the actions, outputs and transitions of the transition system. From context it will be clear what the variables represent.

4-2 Composition

The process of composition when using BDDs is straightforward. The transition boolean function of the composed system is simply the product of all the transition functions of the subsystems:

$$\chi_T((x^n, \dots x^1), (u^n, \dots u^1), (y^n, \dots y^1)) = \bigwedge_i \chi_T^i(x^i, u^i, y^i), \tag{4-2}$$

with a similar expressions for $\chi_{tr}(x)$ and $\chi_B(b, x)$ (which are defined in Section 4-4).

4-3 Synthesization with BDDs

Firstly, an expression for the safe set using boolean functions is constructed. This can be realized by using boolean functions $\chi_{tr}^i(x_i)$ which describe whether state x_i in system S_i is a trigger state, or equivalently if its output is: $H_i(x_i) \in \{T_1, T\}$. The boolean function describing the unsafe is satisfied if two or more of $\chi_{tr}^i(x_i)$ is satisfied. The expression for the safe set is then simply the negation of the unsafe set, resulting in:

$$\chi_W(x) := \neg \bigvee_{i,j \in \{1,\dots,N\},\ i \neq j} \chi_{tr}^i(x_i) \wedge \chi_{tr}^j(x_j), \tag{4-3}$$

Secondly, the safety operator (3-2) can also directly be expressed using a boolean function:

$$\chi_{F_W(Z)}(x) := \chi_W(x) \wedge \exists u.\{[\exists x'.\chi_T(x, u, x')] \wedge [\forall x'.\chi_T(x, u, x') \implies \chi_Z(x')]\}, \tag{4-4}$$

where $\chi_Z(x)$ is the boolean function describing the set Z, and is initialized to $\chi_Z(x) = 1$ before the synthesization iterations.
The iterations are stopped when $\chi_{F_W(Z)}(x) = \chi_Z(x)$. If $\exists x.\chi_Z(x) \neq 0$, a scheduler has been found and a safety controller system can be constructed and represented using boolean function describing its transitions:

$$\chi_{T_c}(x_c, u, x_c') := \chi_T(x_c, u, x_c') \wedge [\forall \alpha.\chi_T(x_c, u, \alpha) \implies \chi_Z(\alpha)]. \tag{4-5}$$

Similarly to before, the scheduler only needs to choose safe inputs from $U_c(x)$, which can be described using the boolean function:

$$\chi_{U_c(x)}(u) := \chi_{U_c}(x, u) = \exists x'.\chi_{T_c}(x, u, x'). \tag{4-6}$$

4-4 Partitioning using BDDs

To describe the blocks in the partition, new boolean variables $b \in \mathbb{B}^{n_b}$ and $c \in \mathbb{B}^{n_b}$ describing the blocks are introduced, as well as a new boolean function:

$$\chi_B(b,x) = \begin{cases} 1 & \text{if} \quad x \in b \\ 0 & \text{otherwise,} \end{cases} \tag{4-7}$$

describing whether state x lies in block b. This boolean function is constructed by using the outputmap H:

$$\chi_B(b,x) = \bigvee_{x \in X, b=H(x)} \bigwedge_i [x_i \iff \sigma_i^x(x)] \wedge [b_i \iff \sigma_i^y(b)))], \tag{4-8}$$

where $\sigma_i^x(\cdot)$ are the encoding functions for the states and $\sigma_i^y(\cdot)$ are the encoding functions for the outputs, which can be directly used for the blocks: $\sigma_i^b(\cdot) := \sigma_i^y(\cdot)$. The transitions are constructed analogously to Equation 3-5:

$$\chi_{T^I}(b,u,c) = \exists x, y. \chi_T(x,u,y) \wedge \chi_B(b,x) \wedge \chi_B(c,y). \tag{4-9}$$

4-5 Refinement using BDDs

Firstly, $Pre(C)$ is expressed with a boolean function using the boolean function $\chi_T(\cdot)$ representing the transition set:

$$\chi_{Pre(C)}(x) = \exists u, x'.[\chi_C(x') \wedge \chi_T(x,u,x')], \tag{4-10}$$

where $\chi_C(x)$ is the boolean function or BDD representing the set C.
A new boolean variable $\beta \in \mathbb{B}$ can be introduced to specify whether $x \in B \cap Pre(C)$ or $x \in B \setminus Pre(C)$, such that $b^+ := (\beta, b^-)$ and

$$\chi_{B^+}(b^+,x) = \chi_{B^+}((\beta,b^-),x) = \chi_{B^-}(b^-,x) \wedge (\beta \oplus \chi_{Pre(C)}(x)). \tag{4-11}$$

The refined partition would then be obtained by iterating over this with all blocks in the current partition, resulting in $\chi_{B^{(i+1)}}(b^{(i+1)},x)$. This is shown in Algorithm 4.2, where $Refine(\cdot)$ is as defined above and $success = True$.

While this efficiently calculates one operation of the refinement operator, this has a large problem. Each time the refinement operator is applied with a block C, a new boolean variable is introduced, effectively causing the number of states that can be represented to be doubled. This causes the representation to be very inefficient as most of the time only a few new states are added. Effectively, the number of boolean variables that represent the blocks are doubled every refinement. This has the effect that at some point performing refinement takes so much time that regular synthesization would have been faster, partly because the number of variables used to represent the blocks far outgrows the number of variables used to represent the states. At this point it would be more useful to perform regular synthesization.
For this another method for refinement in presented in Algorithms 4.1 and 4.2. In short, it loops through all the current blocks and tries to split them. If splitting occurs it adds both blocks to the boolean function, otherwise it will add the original block. After this, it will check if the newly added boolean variables are necessary for the representation, if not it removes them. This is done for all the blocks in the current partition. Finally, if at least one block is split, the new transition function is calculated.

Algorithm 4.1: Refinement operator with block C using BDDs

```
Input: Block variables $b$, $\chi_C(x)$, and $\chi_T(x,u,y)$.
Output: New block variables $b^+$, $\chi_{B^+}(b^+,x)$ and $success$.
$n_b$ is the number of boolean variables $b$;
$b^+$ are new boolean variables with encoding functions $\sigma^+ : \mathbb{N} \to \mathbb{B}^{n_b+1}$;
$\chi_{Pre(C)}(x) \leftarrow \exists u, x'.[\chi_C(x) \wedge \chi_T(x,u,x')]$;
$n \leftarrow 0$;
$\chi_{B^+}(b^+,x) \leftarrow 0$;
for $k \in [0, 2^{n_b}-1]$ do
    $\beta \leftarrow (\sigma_{n_b-1}(k), \ldots, \sigma_0(k))$;
    $\chi_k(x) \leftarrow \chi_B(b,x)|_{b \leftarrow \beta}$;
    $\chi_0(x) \leftarrow \chi_k(x) \wedge \chi_{Pre(C)}(x)$;
    $\chi_1(x) \leftarrow \chi_k(x) \wedge \neg\chi_{Pre(C)}(x)$;
    if $\chi_0(x) = 0 \vee \chi_1(x) = 0$ then
        $\gamma \leftarrow \prod_{j=0}^{n_b}[b_j^+ \iff \sigma_j^+(n)]$;
        $\chi_{B^+}(b^+,x) \leftarrow \chi_{B^+}(b^+,x) \vee (\gamma \wedge \chi_\beta(x))$;
        $n \leftarrow n+1$;
    else
        $\gamma^t \leftarrow \prod_{j=0}^{n_b}[b_j^+ \iff \sigma_j^+(n+t)],\ t \in \{0,1\}$;
        $\chi_{B^+}(b^+,x) \leftarrow \chi_{B^+}(b^+,x) \vee (\gamma^0 \wedge \chi_0(x)) \vee (\gamma^1 \wedge \chi_1(x))$;
        $n \leftarrow n+2$;
        $success \leftarrow True$
if $\chi_{B^+}(b^+,x)|_{b_{n_b}^+ \leftarrow 1} = 0$ then
    $\chi_{B^+}(b^+,x) \leftarrow \exists b_{n_b}^+ . \chi_{B^+}(b^+,x)$;
    Remove $b_{n_b}^+$ from $b^+$;
```

Algorithm 4.2: Refinement using BDDs

```
Input: $\chi_{B^{(i)}}(b^{(i)}, x)$, $\chi_{tr}(x)$, $\chi_T(x, u, y)$
Output: $\chi_{B^{(i+1)}}(b^{(i+1)}, x)$, $\chi_{tr^{(i+1)}}(b^{(i+1)})$ and $\chi_{T^{(i+1)}}(b^{(i+1)}, x, c^{(i+1)})$.
$n_b$ is the number of boolean variables $b^{(i)}$;
$s \leftarrow 0$;
$b^+ \leftarrow b^{(i)}$;
$\chi_{B^+}(b^+, x) \leftarrow \chi_{B^{(i)}}(b^{(i)}, x)$;
for $k \in [0, 2^{n_b} - 1]$ do
    $\beta \leftarrow (\sigma_{n_b-1}(k), \ldots, \sigma_0(k))$;
    $\chi_C(x) \leftarrow \chi_{B^{(i)}}(b^{(i)}, x)|_{b^{(i)} \leftarrow \beta}$;
    $b^+, \chi_{B^+}(b^+, x), success \leftarrow Refine(b^+, \chi_C(x), \chi_T(x, u, y))$ (Algorithm 4.1);
    $s \leftarrow s \lor success$;
if $s$ then
    $b^{(i+1)} \leftarrow b^+$;
    $\chi_{B^{(i+1)}}(b^{(i+1)}, x) \leftarrow \chi_{B^+}(b', x)$;
    $\chi_{tr^{(i+1)}}(b^{(i+1)}) \leftarrow \exists x.\chi_{B^{(i+1)}}(b^{(i+1)}, x) \land \chi_{tr}(x)$;
    $\chi_{T^{(i+1)}}(b^{(i+1)}, x, c^{(i+1)}) \leftarrow \exists x, y.\chi_T(x, u, y) \land \chi_{B^{(i+1)}}(b^{(i+1)}, x) \land \chi_{B^{(i+1)}}(c^{(i+1)}, y)$;
else
    Refinement failed. Thus already maximally refined.
```

4-6 Time Complexity

The main reason for introducing BDDs and the alternative synthesization algorithm is to increase computational efficiency. To study the impact these techniques might have, the time complexity for the algorithms are discussed qualitatively in this section. In the following, N_x denotes the number of states, N_u denotes the number of actions, N_o denotes the number of outputs and N_T denotes the number of transitions for a single TS. For simplicity, $N_T \approx O(N_x^2 \cdot N_u) \approx O(N_x^2)$ (as the number of actions is constant for all TS). The amount of states a the TS scales as $N_x \sim O(k_{max}^2)$, since region $\mathcal{Q}_j$ generates j states in the TS and in worst case there are k_{max} regions. Furthermore, m will denote the number of subsystems.

First consider the time complexity of the safety game without the use of BDDs. Every time the safety operator is applied, it will cause a cost of $O(|X_{comp.}| \cdot |U_{comp.}|) = O(N_x^m) = O(k_{max}^{2m})$ as it calculates $Post_u(x)$ for all states and actions. In the worst case, the safety operator only removes one state every iteration, until the result is the empty set. Thus the time complexity of the safety game is $O(|X_{comp.}| \cdot k_{max}^{2m}) = O(k_{max}^{4m})$.

Secondly, the time complexity of the safety game on partitioned systems without the use of BDDs. For the initial partition, the amount of states for a given TS goes from $O(k_{max}^2)$ to $O(k_{max})$, since the TS contains k_{max} outputs. This means that the safety game has a cost of $O(k_{max}^{2m})$. However, this is only the case if a solution can be found immediately for the first partition. In the worst case, every refinement only splits one block one time, resulting in a total cost for the partitioning and refinement for a single subsystem of $O(N_x \cdot N_T) = O(k_{max}^3)$ (Section 7.3.2 of [8]), with total cost for m subsystems of $O(mk_{max}^3)$. After every refinement, the safety game is performed again, in the worst case with only one more state. The total cost of all the safety games is then

$$O\left(\sum_{i=0}^{k_{max}^2-k_{max}}(k_{max}+i)^{2m}\right)=O\left(k_{max}^{4m+2}\right)$$

This implies that in the worst case, this algorithm scales worse. However, generally only a few refinements are necessary (or even none), resulting in a lower cost on average.

Lastly, the time complexity with the use of BDDs. The number of variables to represent the states is $n_x = log(N_x) = \log k_{max}$ and similar for the actions, blocks, etc. For the safety game, the time complexity can be found by substituting this value: $N_x \to n_x$, resulting in time complexity $O(mk_{max}^{2m}log(k_{max}))$ without partitioning and $O(mk_{max}^{m}log(k_{max}))$ with partitioning. Partitioning has a lesser effect with the use of BDDs, but still scales somewhat better. The problem lies with refinement. In the refinement implementation with BDDs, the refinement operator loops over every block in the current partition and the refinement operator is applied for every block of the old partition, a single refinement will have the cost

$$O\left(\left|X^{(i)}\right|^2[\log\left|X^{(i)}\right|+\log\left|\bar{X}\right|]\right)=O\left((k_{max}+i)^2\cdot[2\log k_{max}+\log(k_{max}+i)]\right)$$

In the worst case, a single refinement will only split a block once until the original system is retrieved, so the total cost will become

$$O\left(\sum_{i=0}^{k_{max}^2-k_{max}}(k_{max}+i)^2\cdot[2\log k_{max}+\log(k_{max}+i)]\right)=O\left(k_{max}^6\log k_{max}\right).$$

Completely refining m subsystems then has the cost $O(mk_{max}^6\log k_{max})$. The total cost for all the safety games in the worst case is then

$$O\left(\sum_{i=0}^{k_{max}^2-k_{max}}m(k_{max}+i)^m\cdot\log(k_{max}+i)\right)=O\left(mk_{max}^{2m+2}\log k_{max}\right).$$

Since refinement without the use of BDDs has a lower cost, an additional algorithm can be proposed where the safety game is solved with the use of BDDs, but refinement is performed without. It is expected that this should scale better, even when accounting for the fact that the BDDs representing the subsystem need to be recomputed after every refinement.

Chapter 5

Scheduling using Late Triggers

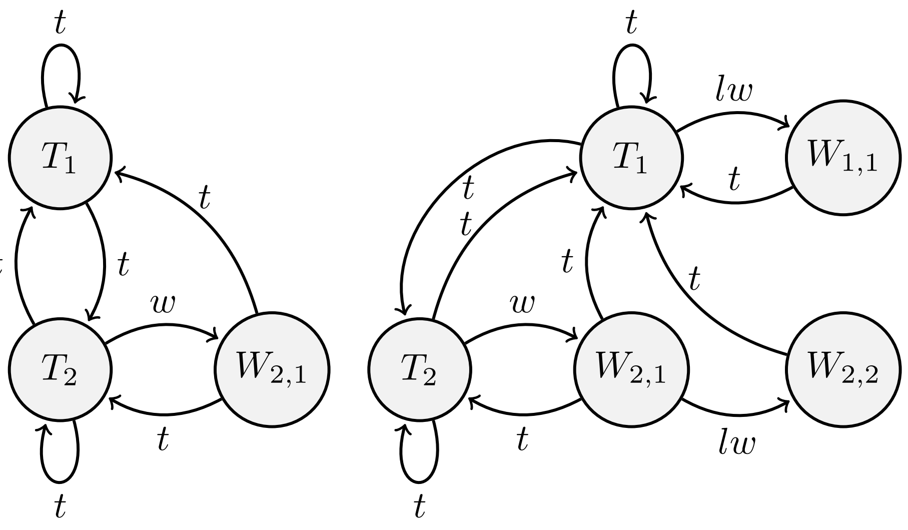
In this chapter another modification to the synthesization algorithm is discussed. In short, the goal of the modification is to allow a certain amount of late triggers in a time period while still retaining stability. Before the details are discussed, first a motivation is given for this approach.

5-1 Motivation

For some collection of subsystem, no scheduler can be found. In this case, one can modify some Periodic Event Triggered Control (PETC) parameters, such as the controller, sampling time or triggering matrix, to try to make a subsystem more schedulable. However, sometimes, this will not work. This can be because of multiple reasons, such as:

- The sampling time cannot be reduced further than it currently is because of physical limitations (e.g. computational or communication limits).
- Changing the triggering mechanism or controller do not affect the nondeterminism of incoming as well as outgoing edges of the state with the lowest triggering time. For a lot of systems this state is the bottleneck regarding scheduling, because the scheduler cannot avoid the transition to this state, causing multiple systems to having to trigger at the same time. Reducing the sampling time considerably could increase the Minimum inter-event time (MIET), however, because of the point above, this cannot be always done.

By allowing late triggers to occur (for a subset of the subsystems), more behaviours are possible, and thus possibly increasing the schedulability of the system. An example of a subsystem which is unschedulable if composed with a copy of itself is shown in Figure 5-1a. In the rest of this chapter, it is discussed how to add late triggers to the models used previously, how to enforce the limitation on the number of late triggers and some considerations regarding stability.



(a) An example unschedulable Transition System (TS) when composed with a copy of itself.

(b) The TS in 5-1a with added late triggers with lateness $L = 1$.

Figure 5-1: An example of a unschedulable TS and its extended form with late triggers.

5-2 Modelling Late Triggers

Late triggers can be added to the TSs by adding states $W_{i,j}$, $j \in [i, i + L)$, where L is the maximum lateness that is allowed. As described in Section 2-2-2, late transitions can also be generated for the traffic model. Previously these have been ignored, but now they can be used. Similar to before, the transitions $(\mathcal{Q}_i, k, \mathcal{Q}_j)$ in $S_{\setminus\mathcal{R}}$ are converted to the TS transitions:

- The states $T_i, W_{i,j}\ j < i$ and the transitions from/to them are the same as in the non-late TS.
- $(W_{i,i}, lw, W_{i,i+1})$, … , $(W_{i,g-2}, lw, W_{i,g-1})$,
- $(W_{i,g-1}, t, T_t)$.

A new action lw ('late wait') has been added, to represent the fact that these transitions are 'late'. An example subsystem with these late triggers is shown in Figure 5-1b.

5-3 Modelling Late Trigger Limitation

Limiting the number of lw actions in a certain time period can be modelled by a counter. Denote by Δ the maximum amount of lw actions, and by r the ratio by how much a lw increases the counter to how much a w or t action decreases the counter. Here assume w and t decrease the counter by 1. The maximum value of the counter is then $r\Delta$, and if it reached more than that, the condition is violated. This counter can be modelled by an auxiliary TS S_{aux}:

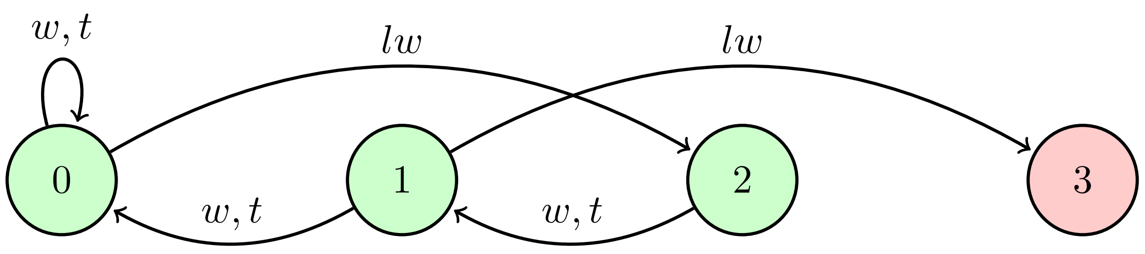


Figure 5-2: An example auxiliary TS with $r = 2$ and $\Delta = 1$.

- $X_{aux} = \{0, \dots, r\Delta, r\Delta + 1\}$
- $U_{aux} := U = \{w, t, lw\}$
- $\xrightarrow[S_{aux}]{} = \{(i, w, max(0, i-1)), (i, t, max(0, i-1)), (i, lw, min(r\Delta + 1, i + r)) \mid i \in X\}$
- $Y_{aux} = \{o, \times\}$
- $H_{aux}(x) = \times$ if $x = r\Delta + 1$, else $H_{aux}(x) = o$.

An example auxiliary system is shown in Figure 5-2, with $r = 2, \Delta = 1$.

This auxiliary system has to be synchronized with the original TS, as both should take the same inputs. This can be achieved by introducing interconnection relation:

$$\mathcal{I} := \{(x, x_{aux}, u, u_{aux} \mid x \in X, x_{aux} \in X_{aux}, u = u_{aux} \in U\}. \tag{5-1}$$

Thus allowing the two system to run in parallel with the exception being that the inputs are equal. The result is a system $\tilde{S}$ which allows for some late triggers to occur, and which keeps track of the amount of late triggers. When using Binary Decision Diagrams (BDDs), this particular interconnection relation can be achieved by using the same input variables for both the original TS as for the auxiliary TS and then using the regular composition (Section 4-2). To synthesize a scheduler for a collection of systems where a subset of subsystem allows for late triggers, the same algorithm as in Chapter 3 can be used, only with modification of the safe set:

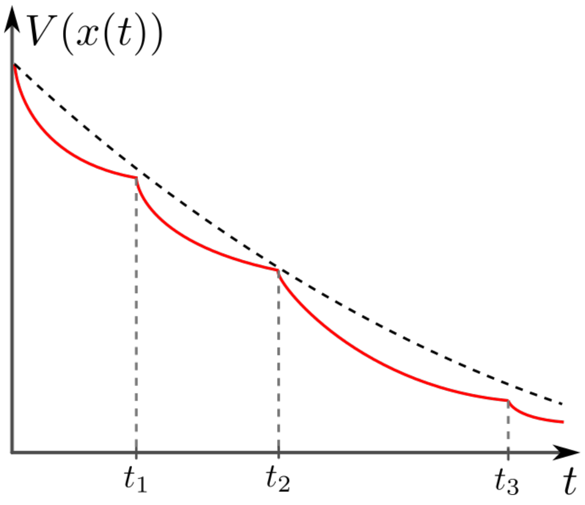
$$W = \{((x_1, x_{aux,1}), \dots, x_n) \in X \mid [\exists_{\leq 1} x_i . H_i(x_i) \in \{T_1, T\}] \wedge [\nexists x_{aux,i} . H_{aux,i}(x_{aux,i}) = \times]\}. \tag{5-2}$$

Or in other words, at most one system can be transmitting at a time **and** no system can trigger late too often.

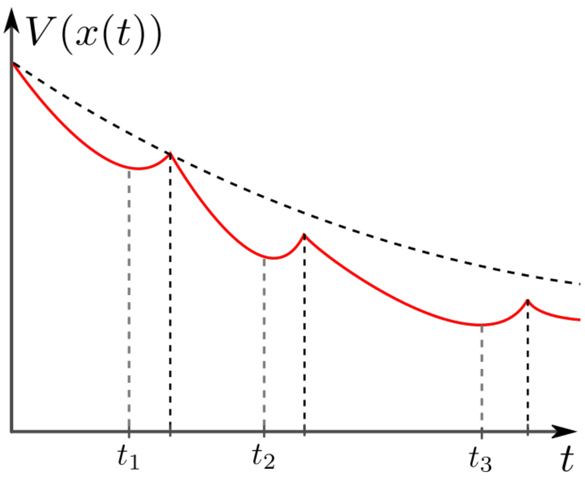
Note that if $\Delta < L$ the last few late states of the extended TS cannot be reached, as such an amount of late triggers cannot occur. This means that these can be removed from the TS, reducing its size somewhat.

5-4 Stability Concerns

A major issue with this approach is that stability guarantees are lost. Stability for a given subsystem is attained when it triggers at or before the point the trigger condition is satisfied,



(a) A typical decrease of a Lyapunov function for a PETC system.



(b) The evolution of a Lyapunov function when allowing late triggers, while still retaining stability. Note that this is not a typical behaviour, and in general it should be assumed that allowing late triggers will cause instability.

which happens when $t - t_i = h\kappa(x(t_i))$ (by definition). However, by the approach above, sometimes a system can trigger later than this, thus losing the stability guarantees. To explain (qualitatively) how stability could be preserved, first the inspiration behind how the limitations on the number of late triggers is discussed.

Consider a LTI system $\dot{x} = Ax + Bu(x)$, with control action $u(x)$ such that the closed loop dynamics are rendered stable. Stability of the closed loop is shown with a Lyapunov function $V(x)$. To convert this into a PETC system, a triggering condition can be designed based on this Lyapunov function by requiring that $\dot{V}(x) \leq -\alpha(|x|)$ for some $\mathcal{K}_\infty$ function α (here $\dot{V}(x)$ is the flow along the new PETC dynamics). A depiction of how the Lyapunov function might evolve for a PETC system is shown in Figure 5-3a. The triggering times correspond to the time instances where the previous inequality become equality. Note that at these time instances, the Lyapunov function is still decreasing.

To take advantage of this, the system might trigger somewhat later than the actual trigger time and still retain stability. This can even be taken a step further: The Lyapunov function can be temporarily allowed to *increase* as long as it decreases overall, for example by bounding it by some function β: $V(x(t)) \leq \beta(x,t)$ which can also act as a Lyapunov function: $\dot{\beta} \leq -\tilde{\alpha}(|x|)$. An example evolution of the Lyapunov function when allowing this behaviour is shown in Figure 5-3b.

The Lyapunov function will decrease overall if the increase caused by late triggers is less than the decrease during normal behaviour. By approximately keeping track of the Lyapunov function ($\tilde{V}$), this can be ensured: $\tilde{V}$ changes by ΔV for each sample before the trigger time and by ΔV_{late} after the trigger time. These values represent the minimal decrease or maximal increase the Lyapunov function can undergo. Instead of these values, the ratio $r := \lceil \frac{\Delta V_{late}}{\Delta V} \rceil$ can be defined, such that the model explained earlier can be used for this process. Note that if ΔV is calculated as:

$$\Delta V := \min_{x} \min_{i \in \{0,\ldots,k_{max}\}} \int_{ih}^{(i+1)h} \dot{V} dt, \tag{5-3}$$

it will result in 0. Similarly for ΔV_{late}, resulting in ∞. This suggests that the ratio r needs to be calculated directly. However, more (formal) analysis on this has to be performed.
Intuitively, this method similar to the approach in [17]. Here an auxiliary variable is introduced which models the Lyapunov function. The triggering condition is then modified using this variable, such that the Lyapunov function is allowed to increase, as long as on average it decreases at least with some rate. In [18], the triggering condition is relaxed even further, such that the actual decay of the Lyapunov function matches much more closely to the desired rate. These approaches have the advantage of decreasing the amount of transmissions needed (thus possibly increasing schedulability).

Chapter 6

Implementation

This chapter will discuss shortly how the algorithms presented are implemented and globally how the resulting schedulers can be implemented and used for real systems and some challenges that may arise when doing this.

6-1 Algorithm Implementations

The algorithms presented in this thesis are implemented using Python. The abstraction algorithm from Section 2-2-2 is implemented, by the author of [6], in the SENTIENT toolbox[1]. Since the tool is still in development, an older version of the toolbox is used. The implementation of the algorithms discussed in this thesis is currently being integrated into this toolbox, so it is not available yet. However, in this section,

The implementation does not have any dependencies on third-party packages except for the package *dd* [19], which is used to implement the Binary Decision Diagram (BDD) parts of the algorithm. An advantage of this package is that it has optional bindings to the C package CUDD [20], which drastically speed up performance. The structure of the Python implementation is for the most part split into two: One implementation without the use of BDDs, and one with the use of BDDs. This mostly pertains to the classes representing the single control loops and their variants which allow late triggers, and the classes which compose multiple control loops into a single system. The steps that are taken to synthesize a scheduler are as follows:

1. First specify multiple systems with dynamics matrices, a controller that renders the closed loop stable and triggering matrices that convert the systems into the Periodic Event Triggered Control (PETC) form.

2. These are then one by one converted into traffic models by the SENTIENT tool.

[1]As of writing not yet released.

3. Each of the resulting traffic models is converted to a Transition System (TS) by the class '*nfa*' or '*nfa_late*' when late triggers should be used. The class '*nfa_late*' has three more arguments representing the parameters r, Δ and L.

4. All the resulting objects representing the TSs are combined into a list and from this one of the following is created: '*nfa_system*', '*nfa_system_bdd*', '*nfa_late_system*' or '*nfa_late_system_bdd*'. These objects represent the composed systems. However, the systems are not actually composed until the '*compose()*' method is called.

5. One of the following functions is then called with this object (the first two allow for all four types of systems):

    - '*SynthesizeSafetySchedulerSimple()*': This implements the standard synthesization algorithm. All four types of systems can be supplied.
    - '*SynthesizeSafetySchedulerPartitioningAll()*': This implements the synthesization algorithm which makes use of partitioning (Algorithm 3.2). All four types of systems can be supplied.
    - '*SynthesizeSafetySchedulerPartitioningViaNFA()*': This implements the same algorithm as above, however, this time the refinement is performed without BDDs.

6. The output is either a Python dictionary or a BDD representing $U_c(x)$ when a scheduler has been found, or *None* or $\chi(x, u) = 0$ when the scheduling problem is infeasible.

An example code snippet which implements one of the scheduling algorithms is shown in Figure 6-1. Another code snippet showing the safety game implementation using BDDs is shown in Figure 6-2.

6-2 Issues Regarding Scheduler Implementation

Physical implementations of the one of the schedulers is quite straightforward. Every sample, the scheduler chooses whether to trigger one of the subsystems and which, and depending on that action, the next states of the subsystems are updated. If a subsystem has been triggered, the state-space region is it currently in is recomputed, and the corresponding state in its TS is updated. To calculate the next state the scheduler needs two additional sources information:

- The region descriptors, which tell whether a continuous state x lies in a traffic model state: $x \in \mathcal{Q}_k$. These are necessary to update the TS state when a subsystem triggers.

- The individual TSs of the subsystems, to determine the next state when a subsystem waits.

The process is described in Algorithm 6.1.

Algorithm 6.1: Scheduler

```
Input: List of systems $S_i$, TS states $s_i$ and state space states $x_i$.
Output: List of actions $u$ for each of the subsystems, next TS state $s^+$.
Take any $a \in U_c(s)$;
$u \leftarrow a$;
for $s_i \in s, u_i \in u$ do
    if $u_i = t$ then
        $s_i^+ \leftarrow UpdateRegion(x_i)$;
    else
        Take any $a \in Post_w(s_i)$;
        $s_i^+ \leftarrow x$;
    end
end
```

Recalculating the region every trigger event might not necessary, however, this does improve the disturbance rejection as well as modelling inaccuracies. To save computation, the region can be re-calculated less often with the risk of losing stability. For example, suppose a system is in region $\mathcal{R}_i$ and triggers. If the next state is based on its TS, it could happen that it lands in a state (because of nondeterminism) which has a higher allowable inter-event time than what is allowed in the real region the system is in, causing it to trigger possibly too late. If this occurs consistently enough, the system might become unstable.

Instead of the scheduler taking any action from $U_c(s)$, it could first check if $(w, \dots, w) \in U_c(s)$ and take that action if possible. This enforces the systems to wait to trigger as long as possible, lowering the average inter-event time. While this is generally not the optimal strategy, it still offers a high potential performance gain with minimal logic. A comparison by means of simulation between taking a random action and prioritizing $(w, \dots, w)$ is made in Section 7-2-1.

When implemented using a dictionary, the scheduler basically becomes a look-up table, and thus choosing an input is $O(1)$. However, when the scheduler is generated using BDDs, choosing an input is different. Instead of acting as a look-up table, the scheduler only tells whether a given input is valid or not. A solution to this is to convert the BDD representation of the scheduler to the dictionary representation. However, this loses the relative compact size of the scheduler as opposed to the size of the dictionary which can very well be too huge to handle effectively (See Section 7-1-3 for a memory usage comparison).

The better solution is to loop over different input values and check if they are valid. If done naively, this process is $O(n_u^m)$. However, it is known beforehand that only one system can trigger at once, thus choosing an input becomes $O(m+1)$ without late triggers and $O((1+m-l/2)\cdot 2^l)$ with late triggers, where l is the number of subsystem which can trigger late. These values can be deduced by counting possibly allowed inputs. Efficiency can be improved even further by utilizing the techniques from [21] (which focuses on symbolic controllers, but is applicable to this as well). Here multiple copies of the same BDD are implemented on a FPGA, but with one or multiple input binary variables are already set (and the resulting BDDs reduced). Then instead of looping through all possible values, only the output of these BDDs needs to be checked to determine a valid input.

```python
def SynthesizeSafetySchedulerPartitioningAll(S):
    """
    Tries to synthesize a scheduler somewhat more efficiently by reducing the size all subsystem,
    and synthesizing on the composition of those. If it fails, it will refine each subsystem and
    try again, until refinement is no longer possible.
    :param S:
    :return:
    """
    print("Starting scheduler synthesization.")
    print("Partitioning all subsystems.")
    res = S.partition_all()
    if not res:
        print("Partitioning failed.")
        return None

    num_tries = 1
    RefSuccess = True
    # Start synthesization loop
    while RefSuccess:
        print("Try: {n}".format(n=num_tries))
        print("Composing system.")
        S.compose()

        print("Generating safe set.")
        W = S.safe_set()
        if W is None:
            print("No safe set found.")
            return None

        print("Solving safety game.")
        Z = S.safety_game(W)
        if Z is None or (type(S) is nfa_system_bdd and Z == S.bdd.false):
            tempSuccess = False
            print("No solution to safety game found.")
            print("Refine all subsystem and try again..")
            RefSuccess = S.refine_all()
            num_tries += 1
            continue

        print('Safety game solved!')
        print('Generating scheduler.')
        C = S.create_controller(Z, StatesOnlyZ=False)
        return C

    print('Refinement no longer possible.')
    print('No scheduler found.')
```

Figure 6-1: Code snippet implementing the alternative scheduler synthesis algorithm.

```python
def safety_game(self, W: _bdd.Function = None):
    """
    Solves the safety game for given safety set W (expressed as BDD
        function)
    :param W: BDD function defining the safety set
    :return: BDD function defining the solution Z of the safety game
    """
    if W is None:
        W = self.safe_set()

    rename = {i:j for (i,j) in zip(self.bvars, self.cvars)}
    Z_new = self.bdd.true
    Z_old = self.bdd.false
    while Z_old != Z_new:
        Z_old = Z_new
        Z_r = self.bdd.let(rename, Z_old)
        Z_new = self._safety_operator(W, Z_r)
    return Z_new

def _safety_operator(self, W: _bdd.Function, Z: _bdd.Function):
    B1 = self.bdd.exist(self.cvars, self.tr)
    B2 = self.bdd.forall(self.cvars, self.bdd.apply('->', self.tr, Z))
    B3 = self.bdd.apply('&', B1, B2)
    Z_new = self.bdd.apply('&', W, self.bdd.exist(self.uvars, B3))
    return Z_new
```

Figure 6-2: Code snippet of two methods from the class '*nfa_system_bdd*' implementing the safety game using BDDs.

Chapter 7

Benchmarks and Simulations

7-1 Benchmarks

Several benchmarks have been performed to compare the performance of the different algorithms. For clarity in the figures, the algorithms have been numbered as follows:

1. Alg. 1: Regular Synthesization, without Binary Decision Diagrams (BDDs).
2. Alg. 2: Synthesization with partitioning and refinement, without BDDs.
3. Alg. 3: Regular Synthesization, with BDDs.
4. Alg. 4: Synthesization with partitioning and refinement, with BDDs.
5. Alg. 5: Synthesization with BDDs, but partitioning and refinement without.

7-1-1 Deterministic Subsystems

First, the multiple synthesization methods are benchmarked with identitical deterministic subsystems. These subsystem have the property that if the maximum trigger time is T, and N subsystems are composed, a scheduler can always be found if $N \leq T$. The tests are performed for $N = 2$ and $N = 3$ with and without partitioning and BDDs. In the case of BDDs, two different partitioning methods are also tested: partitioning using BDDs and regular partitioning with TSs. The results are shown in Figures 7-1and 7-2. Additionally, the synthesization time versus the number of subsystems is also shown in Figure 7-3. In this case, only Algorithms 3 are 4 are shown, as synthesis without using BDDs grows too quickly with the number of subsystems, and Algorithm 5 has a negligible difference with Algorithm 3. For $m = 2$ subsystems, using no BDDs and partitioned subsystems is the fastest. This is likely caused by the overhead of representing the Transition Systems (TSs) by BDDs. However, when $m = 3$, using BDDs is much faster, and when $m = 4$, only a few measurements did not time out for

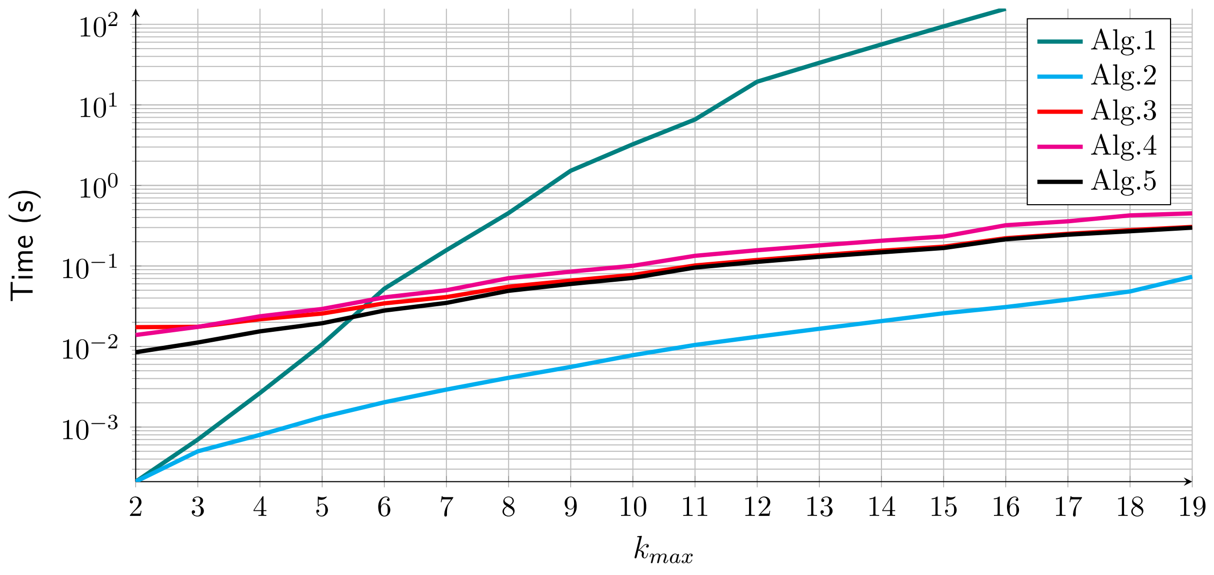


Figure 7-1: Computation time of synthesizing a scheduler (excluding the creation of abstractions) for deterministic systems (as described in Section 7-1-1) with $m = 2$ subsystems.

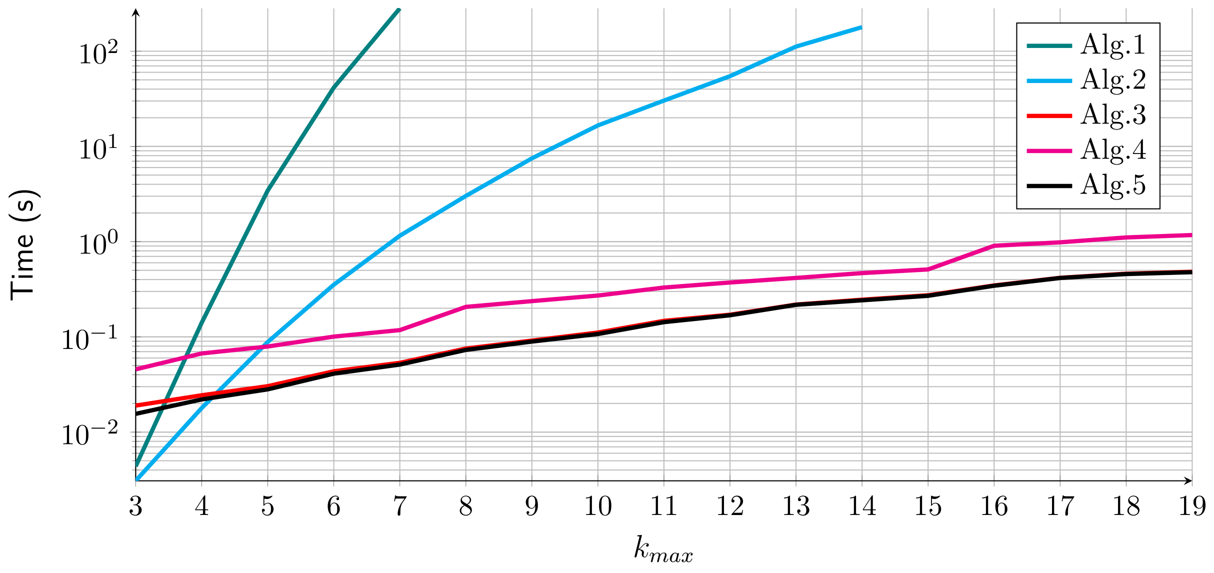


Figure 7-2: Computation time of synthesizing a scheduler (excluding the creation of abstractions) for deterministic systems (as described in Section 7-1-1) with $m = 3$.

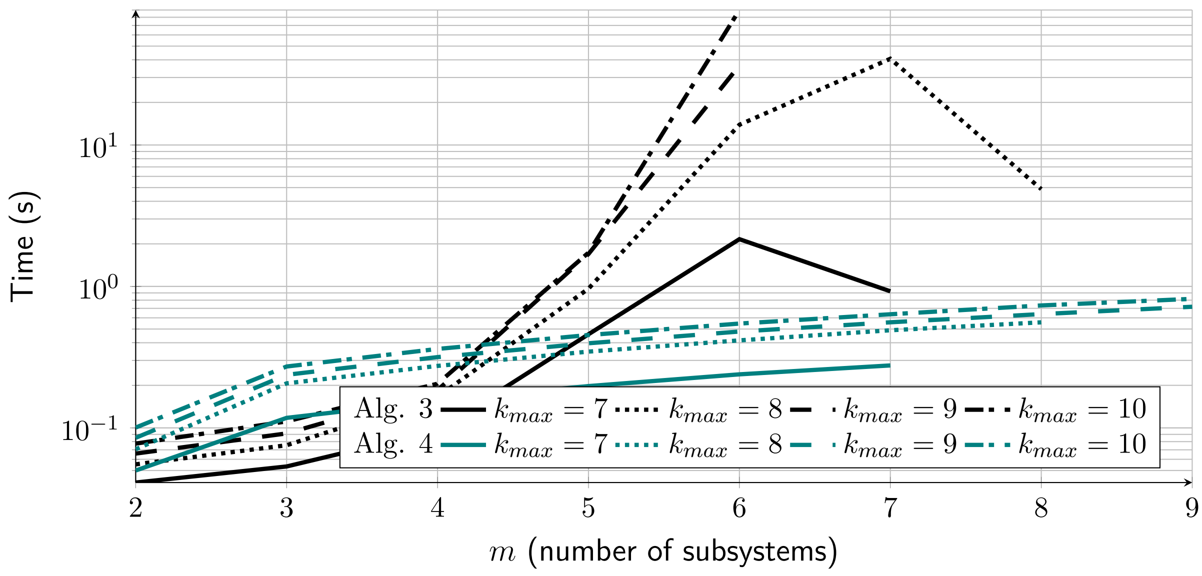


Figure 7-3: Computation time of synthesizing a scheduler (excluding the creation of abstractions) for deterministic systems versus the number of subsystems used.

7-1-2 Random Subsystems

The second set of benchmarks measures the time for random systems, but this time only using BDDs. For each measurement, first it is check whether the random system is schedulable, if not a new system is generated, otherwise the system is used for measurement for all three techniques. This ensures that the comparison is fair. The results are shown in Figures 7-4 and 7-5.

7-1-3 Auxiliary Benchmarks

Two more benchmarks have been performed to study scalability. Firstly, how the size of the subsystems impacts the number of iterations needed to solve the safety game. This is shown in Figure 7-6. The size of each subsystems is expressed with the number of transitions. This result is interesting, as it shows that the number of iterations decreases after a certain point. Secondly, the memory used to represent the composed systems, with and without partitioning and with and without BDDs. This is shown in Figures 7-7 and 7-8. Without BDDs, the memory use scales poorly especially with the number of subsystems and somewhat with the maximum trigger time. However, partitioning decreases the memory use by a lot, implying better scalability. Using BDDs lowers the memory usage to comparatively negligible amount, and partitioning in this case even almost doubles the memory usage.

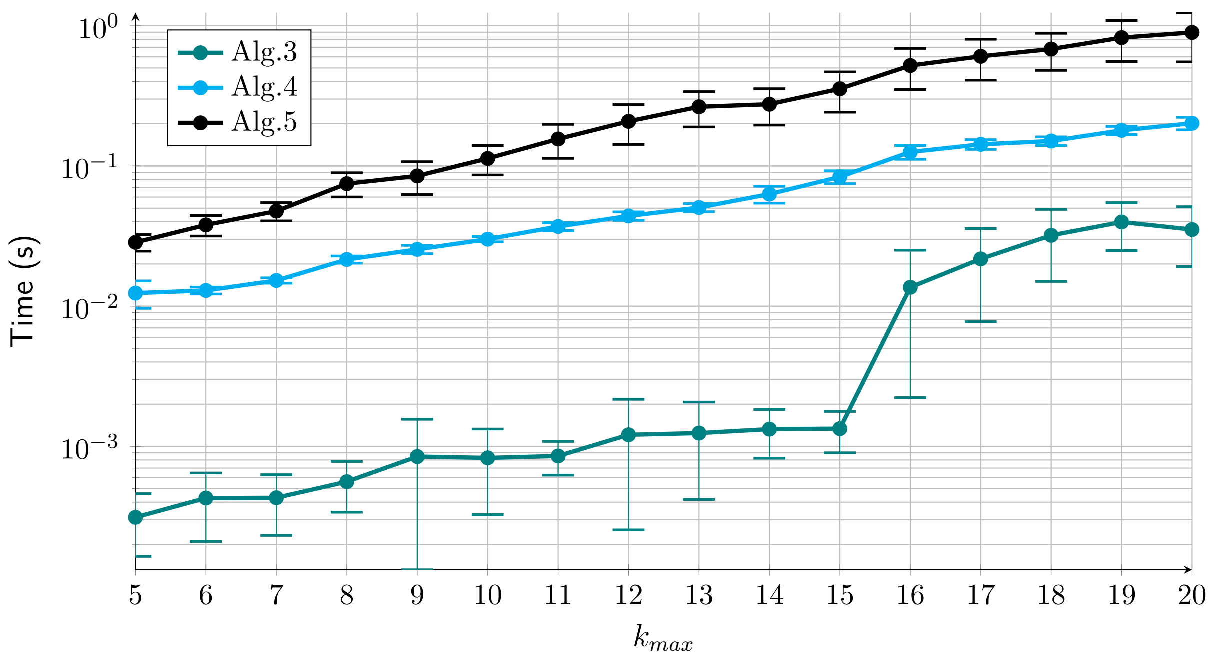


Figure 7-4: Computation time of synthesizing a scheduler (excluding the creation of abstractions) for random system (as described in Section 7-1-2) with $m = 2$ subsystems. The error bars represent one standard deviation.

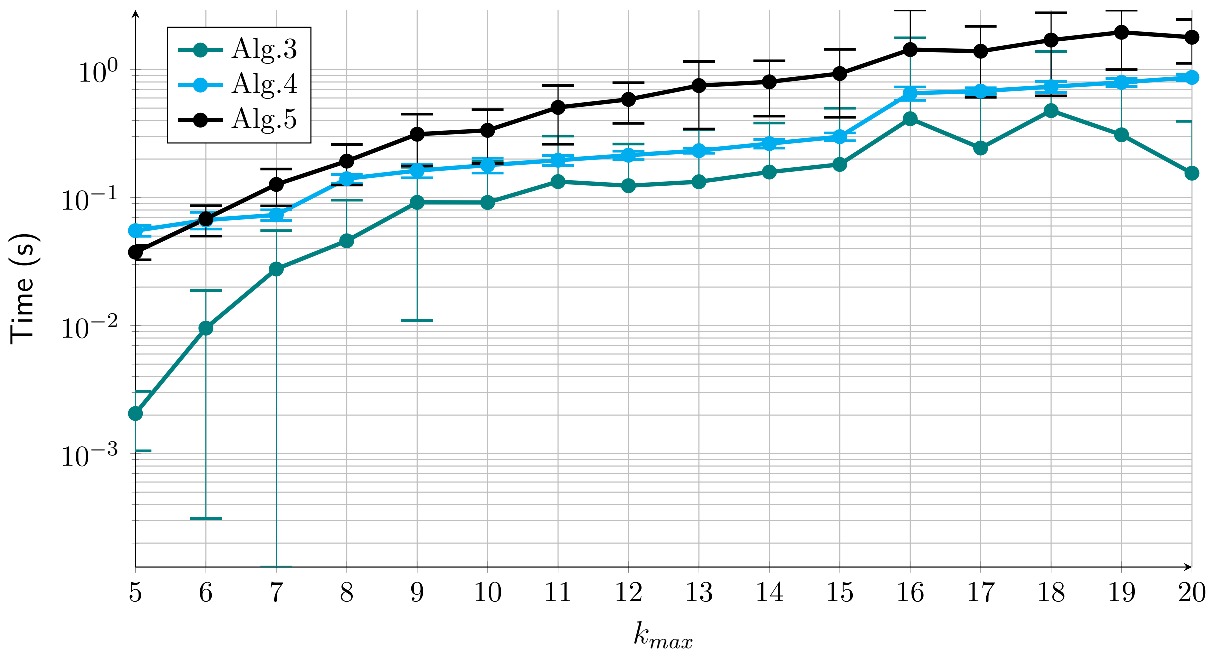


Figure 7-5: Computation time of synthesizing a scheduler (excluding the creation of abstractions) for random system (as described in Section 7-1-2) with $m = 3$ subsystems. The error bars represent one standard deviation.

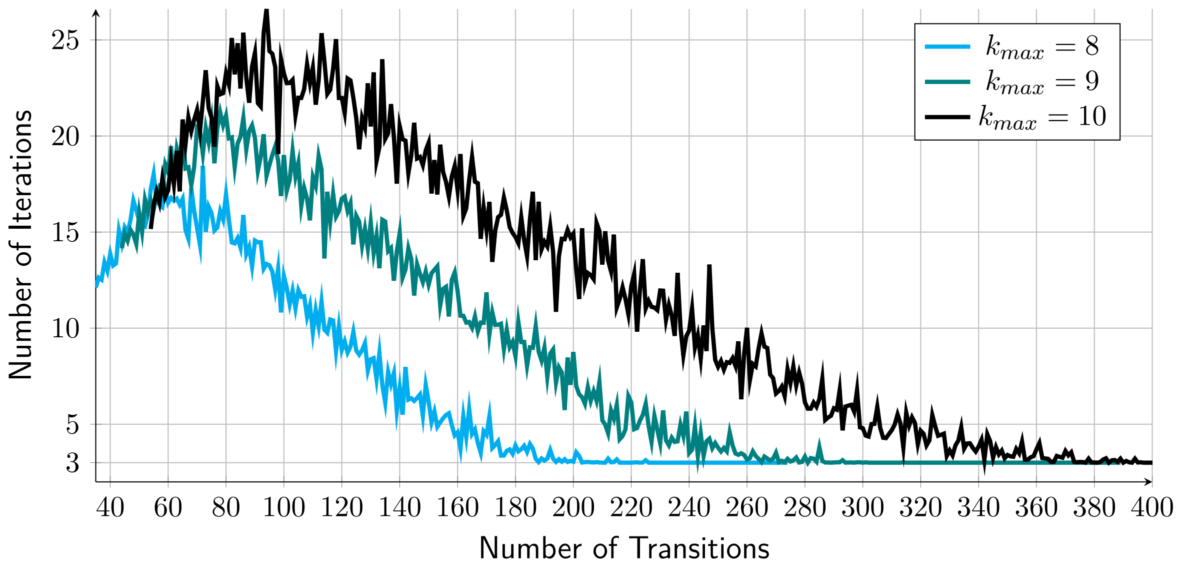


Figure 7-6: Number of iterations needed to solve the safety game versus the number of transitions in each subsystem. Here $m = 2$ subsystems are used. The minimum number of transitions is such that the transition system for the subsystems stay connected.

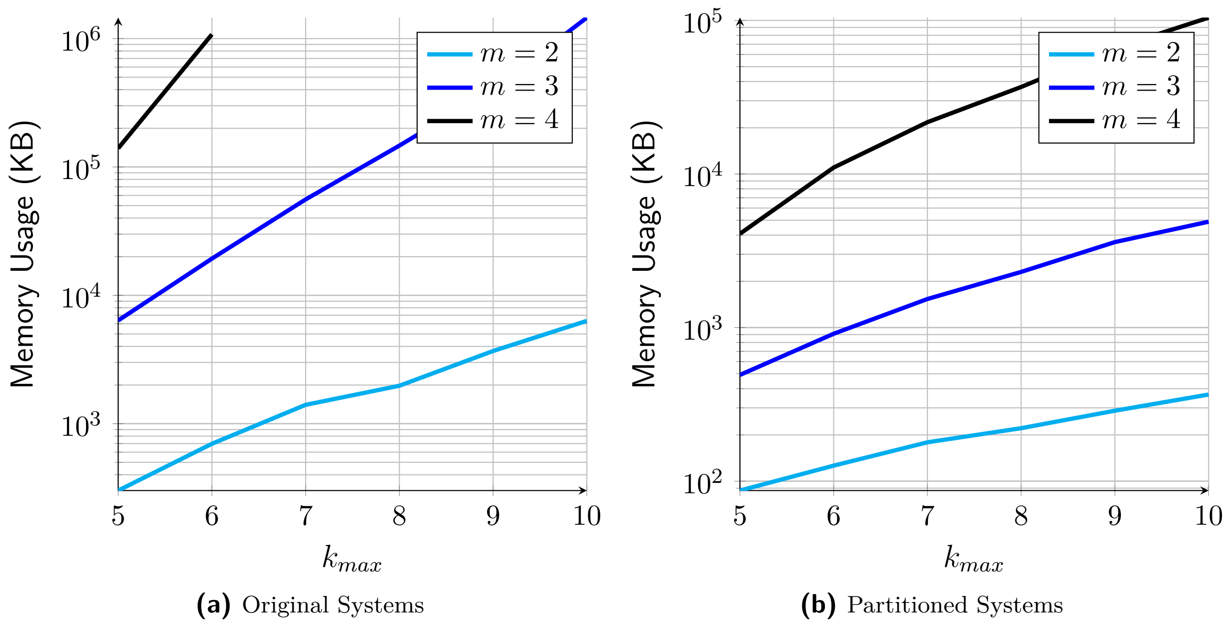
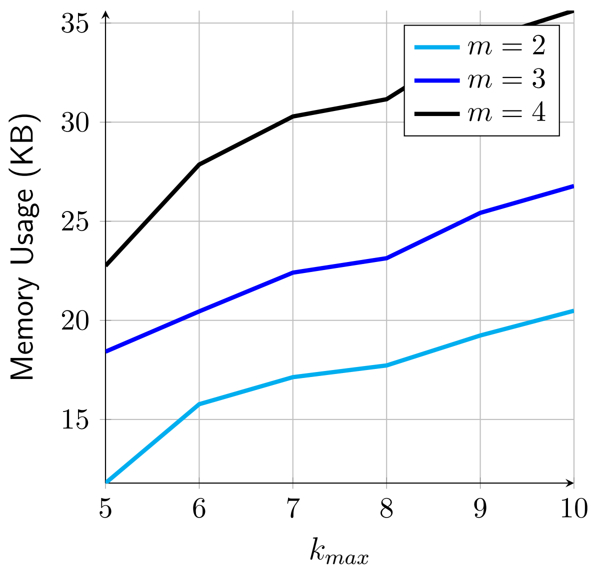
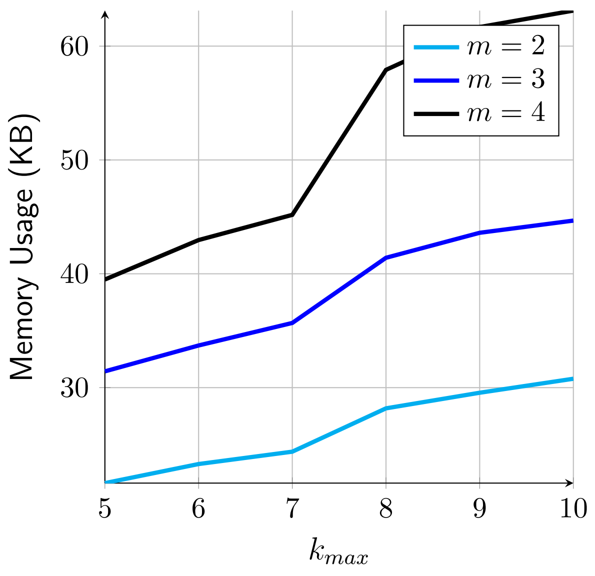


Figure 7-7: Memory usage (to represent the systems) for $m = 2, 3, 4$ subsystem without the use of BDDs.



(a) Original Systems



(b) Partitioned Systems

Figure 7-8: Memory usage (to represent the systems) for $m = 2, 3, 4$ subsystem with the use of BDDs.

7-2 Simulation of Linearized Batch Plants

Consider the plants [22]:

$$\dot{\xi}_i(t) = \begin{bmatrix} 1.38 & -0.208 & 6.715 & -5.676 \\ -0.581 & -4.29 & 0 & 0.675 \\ 1.067 & 4.273 & -6.654 & 5.893 \\ 0.048 & 4.273 & 1.343 & -2.104 \end{bmatrix} \xi_i(t) + \begin{bmatrix} 0 & 0 \\ 5.679 & 0 \\ 1.136 & 3.146 \\ 1.136 & 0 \end{bmatrix} u_i(t), \quad i \in \{1, 2\} \tag{7-1}$$

Following the same procedure as in [6], controllers and triggering mechanisms are generated as follows. The two controllers are constructed using LQR with the matrices $Q_1 = Q_2 = I$ and $R_1 = 0.2I$, $R_2 = 0.1I$, resulting in the controllers:

$$\begin{aligned} u_1(t) &= \begin{bmatrix} 0.52 & -1.97 & -0.45 & -2.14 \\ -3.81 & -0.023 & -2.80 & 1.67 \end{bmatrix} \xi_1(t) = K_1 \xi_1(t) \\ u_2(t) &= \begin{bmatrix} 0.91 & -2.73 & -0.56 & -3.07 \\ -4.82 & 0.0059 & -3.63 & 2.04 \end{bmatrix} \xi_2(t) = K_2 \xi_2(t) \end{aligned} \tag{7-2}$$

The triggering conditions are based on Lyapunov function such that they guarantee that

$$\dot{V}(t) \leq -\rho_i \xi_i(t)^T P_i \xi_i(t), \tag{7-3}$$

in this case P_i are the solutions of the continuous time Lyapunov equation with matrix $Q_{lyap,i} = Q_i + K_i^T R_i K_i$ and $\rho_1 = \rho_2 = 0.8$. From the condition 7-3 the triggering matrices are generated, resulting in:

$$\begin{aligned} \Psi_1 &= \begin{bmatrix} 5.13 & -0.34 & 3.75 & -2.69 & -2.96 & 0.19 & -2.09 & 1.5 \\ -0.34 & 1.2 & 0.34 & 1.5 & 0.19 & -0.78 & -0.19 & -0.84 \\ 3.75 & 0.34 & 2.69 & -1.34 & -2.09 & -0.19 & -1.6 & 0.74 \\ -2.69 & 1.5 & -1.34 & 2.45 & 1.5 & -0.84 & 0.74 & -1.47 \\ -2.96 & 0.19 & -2.09 & 1.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.19 & -0.78 & -0.19 & -0.84 & 0.0 & 0.0 & 0.0 & 0.0 \\ -2.09 & -0.19 & -1.6 & 0.74 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.5 & -0.84 & 0.74 & -1.47 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \\ \Psi_2 &= \begin{bmatrix} 4.13 & -0.45 & 3.05 & -2.27 & -2.4 & 0.25 & -1.7 & 1.26 \\ -0.45 & 1.15 & 0.27 & 1.51 & 0.25 & -0.75 & -0.15 & -0.84 \\ 3.05 & 0.27 & 2.23 & -1.02 & -1.7 & -0.15 & -1.35 & 0.57 \\ -2.27 & 1.51 & -1.02 & 2.24 & 1.26 & -0.84 & 0.57 & -1.36 \\ -2.4 & 0.25 & -1.7 & 1.26 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.25 & -0.75 & -0.15 & -0.84 & 0.0 & 0.0 & 0.0 & 0.0 \\ -1.7 & -0.15 & -1.35 & 0.57 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.26 & -0.84 & 0.57 & -1.36 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \end{aligned} \tag{7-4}$$

Constructing the abstractions with sampling time $h = 0.01$ and $k_{max,1} = k_{max,2} = 20$, results in traffic models with regions:

$$\begin{aligned} \mathcal{R}_1 &= \{\mathcal{Q}_8, \ldots, \mathcal{Q}_{20}\} \\ \mathcal{R}_2 &= \{\mathcal{P}_6, \ldots, \mathcal{P}_{20}\} \end{aligned} \tag{7-5}$$

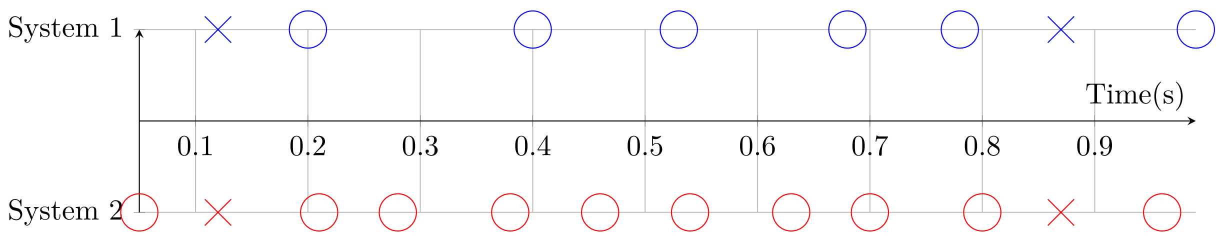
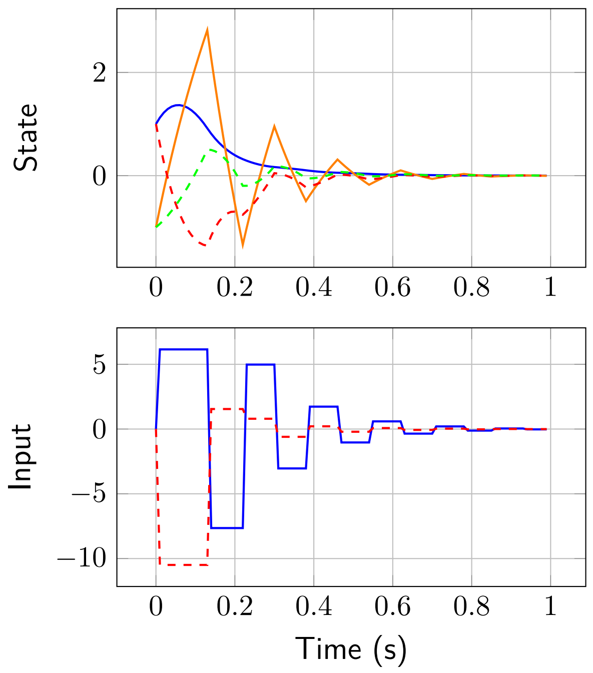
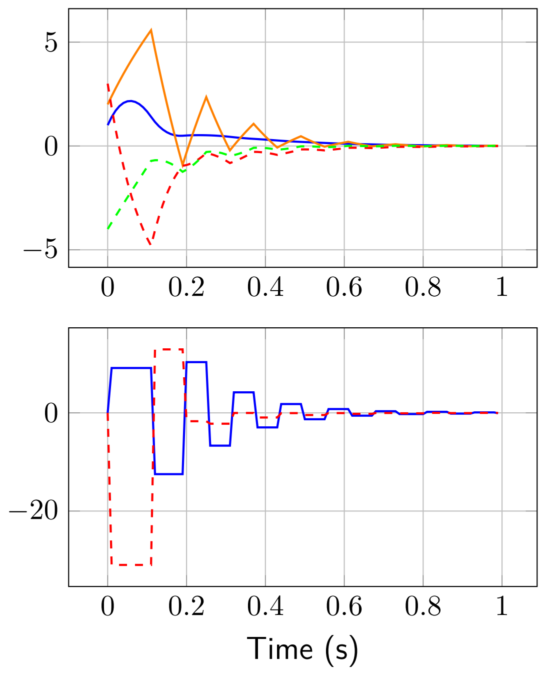


Figure 7-9: Resulting triggering times of a simulation of the systems 7-1 without the use of a scheduler. Triggers are denoted with an 'O'. When both systems trigger simultaneously, the trigger event is denoted with an 'X' instead.

A scheduler can be found for these traffic models. A simulation of the plants with the scheduler is shown in Figure 7-10, how the traffic model regions evolve is shown in Figure 7-11 and the triggering instants are shown in Figure 7-12. For reference, a simulation of both system without the use of a scheduler has been performed as well, resulting in the triggering events as shown in Figure 7-9. The unscheduled subsystems have two collisions of their triggering events. With the introduction of a scheduler these collisions have been successfully avoided while retaining stability.



(a) Simulation of the plant 7-1 with controller K_1 (7-2) with initial condition $\xi_0 = [1, -1, 1, -1]^T$, $h = 0.01$ and $k_{max} = 20$.



(b) Simulation of the plant 7-1 with controller K_2 (7-2) with initial condition $\xi_0 = [1, 2, 3, -4]^T$, $h = 0.01$ and $k_{max} = 20$.

Figure 7-10

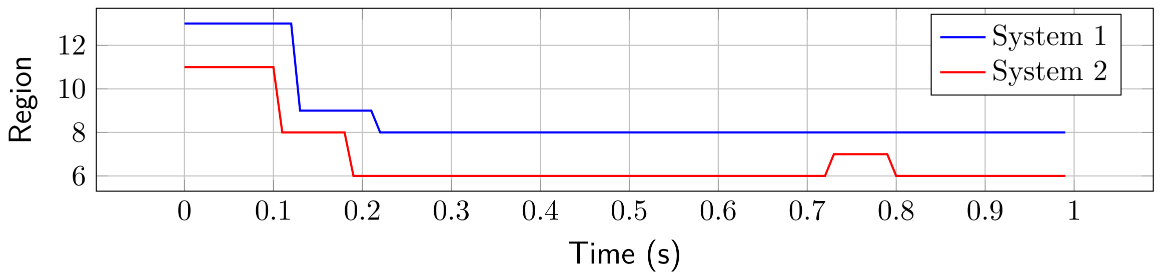


Figure 7-11: The traffic model regions the systems are in over time, corresponding to Figure 7-10.

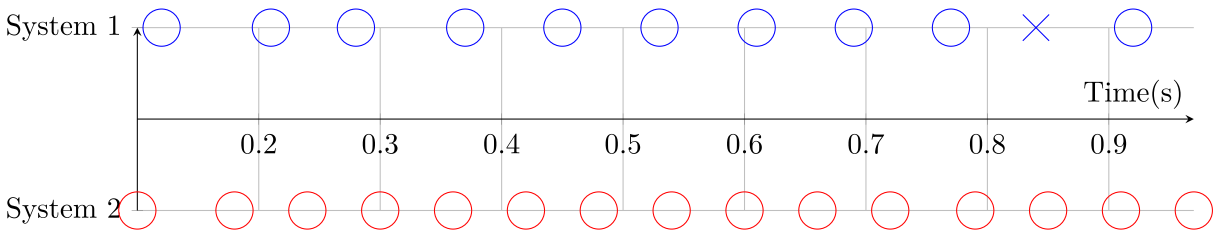
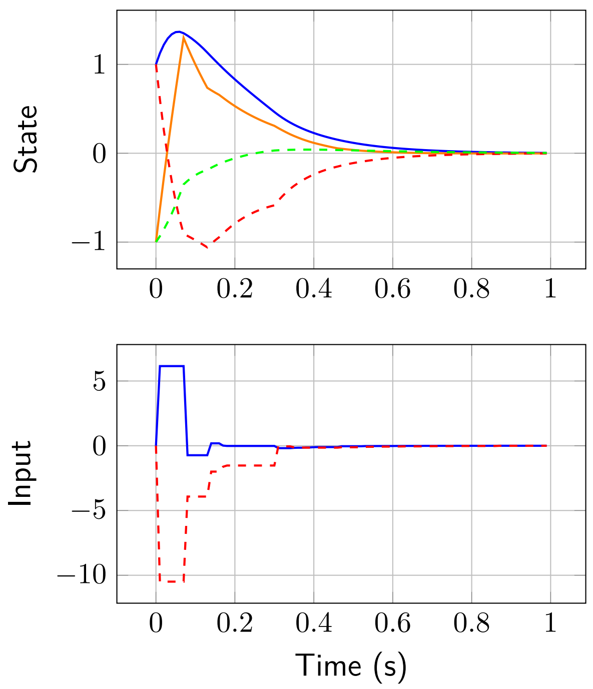
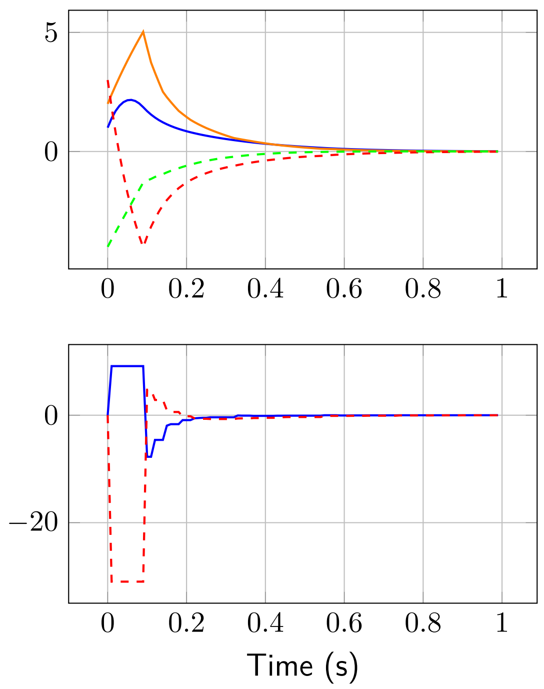


Figure 7-12: Triggering Times for both systems corresponding to Figure 7-10. Early triggers are denoted by a 'X', and triggers on the deadline are denoted with a 'O'.



(a) Simulation with 'random' inputs of the plant 7-1 with controller K_1 (7-2) with initial condition $\xi_0 = [1, -1, 1, -1]^T$, $h = 0.01$ and $k_{max} = 20$.



(b) Simulation with 'random' inputs of the plant 7-1 with controller K_2 (7-2) with initial condition $\xi_0 = [1, 2, 3, -4]^T$, $h = 0.01$ and $k_{max} = 20$.

Figure 7-13

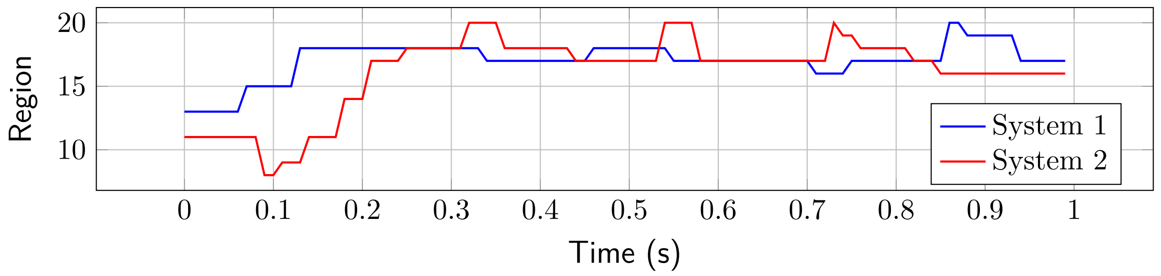


Figure 7-14: The traffic model regions the systems are in over time, corresponding to Figure 7-13

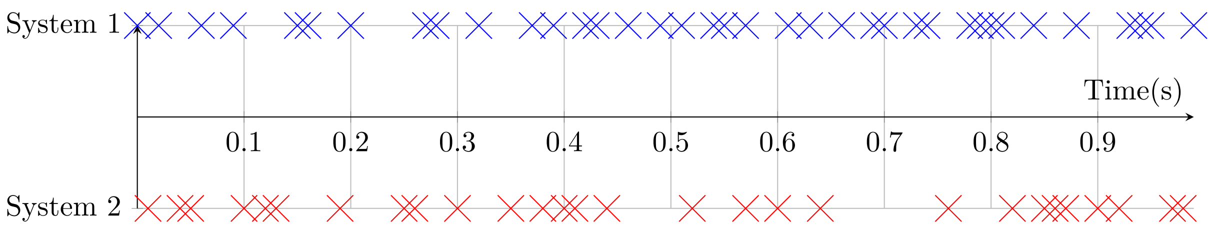


Figure 7-15: Triggering Times for both systems corresponding to Figure 7-13. Early triggers are denoted by a 'X', and triggers on the deadline are denoted with a 'O'.

7-2-1 Simulation with Randomized Inputs

In the previous simulation, if the action $(w, \ldots, w)$ was possible, it would be taken. To illustrate the advantage this has over taking a random (but safe) action, simulation with random inputs are shown in Figures 7-13, 7-14 and 7-15. Random inputs result in a much higher transmission rate. Additionally, for this simulation every trigger is early, as opposed to the earlier results where only one early trigger occurred.

7-3 Simulation with late triggers

First consider the system [23]:

$$\begin{aligned} \dot{\xi}_1(t) &= \begin{bmatrix} 0 & 1 \\ -2 & 3 \end{bmatrix} \xi_1(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_1(t), \text{ and} \\ u_1(t) &= \begin{bmatrix} 1 & -4 \end{bmatrix} \xi_1(t). \end{aligned} \tag{7-6}$$

and the second system [24]:

$$\begin{aligned} \dot{\xi}_2(t) &= \begin{bmatrix} -0.5 & 0 \\ 0 & 3.5 \end{bmatrix} \xi_2(t) + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u_2(t), \text{ and} \\ u_2(t) &= \begin{bmatrix} 1.02 & -5.62 \end{bmatrix} \xi_2(t). \end{aligned} \tag{7-7}$$

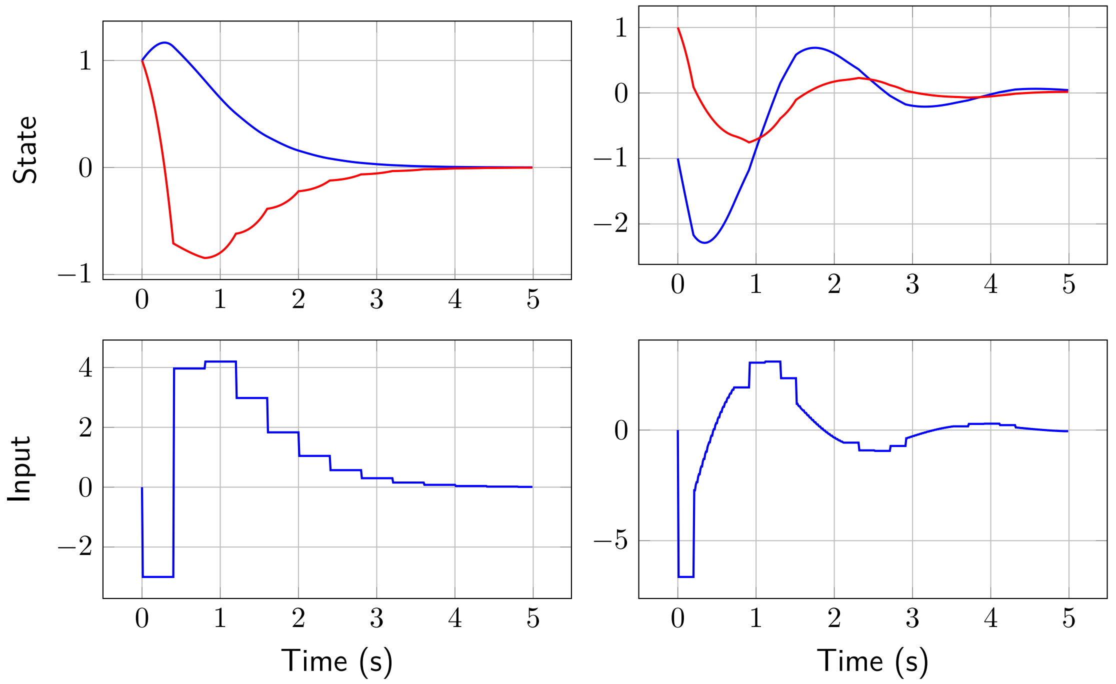
Both with sampling time $h = 0.01$ and maximum inter-event time $k_{max,1} = 40$ and $k_{max,2} = 20$. Using the same techniques as above to generate the triggering matrices results in:

$$\begin{aligned} \Psi_1 &= \begin{bmatrix} -1.95 & 1 & 0.5 & -2 \\ 1 & 7.05 & 1 & -4 \\ 0.5 & 1 & 0 & 0 \\ -2 & -4 & 0 & 0 \end{bmatrix} \\ \Psi_2 &= \begin{bmatrix} 3.41 & -6.76 & -0.86 & 4.72 \\ -6.76 & 13.15 & 2.51 & -13.84 \\ -0.86 & 2.51 & 0 & 0 \\ 4.72 & -13.84 & 0 & 0 \end{bmatrix} \end{aligned} \tag{7-8}$$

Constructing abstractions of these systems results in traffic models with regions:

$$\begin{aligned} \mathcal{R}_1 &= \{\mathcal{Q}_{38}, \mathcal{Q}_{39}, \mathcal{Q}_{40}\} \\ \mathcal{R}_2 &= \{\mathcal{P}_1, \ldots, \mathcal{P}_{20}\} \end{aligned} \tag{7-9}$$

Unfortunately, using the regular synthesization algorithm, no scheduler can be found. The likely explanation for this is that the second system has a lot of nondeterminism around $\mathcal{Q}_1$ (in- and outgoing edges). Using late triggering however, a scheduler can be found. Allow the second system to have late triggers with $\Delta = 1$ and $r = 2$. Results from simulations are shown in Figure 7-16. How the traffic model regions change over time is shown in Figure 7-17. While stability with these late triggers is not proven, it seems that (at least with these initial conditions), the systems will stabilize. Notable is that the triggering times (shown in Figure 7-18) for the second mostly occur in bursts. This is because the system is 'stuck' in $\mathcal{P}_1$, as is also shown in Figure 7-17. It should be noted that in [3] a scheduler has been successfully found using different triggering mechanisms. Thus as reference, a scheduler has also been synthesized with those triggering mechanisms, and a simulation has been performed. The results of this are shown in the Appendix A.



(a) Simulation of the plant 7-6 with initial condition $\xi_0 = [1, 1]^T$, $h = 0.01$ and $k_{max} = 40$.

(b) Simulation of the plant 7-7 with initial condition $\xi_0 = [1, -1]^T$, $h = 0.01$, $k_{max} = 20$ and late triggers with $\Delta = 1$, $r = 2$.

Figure 7-16

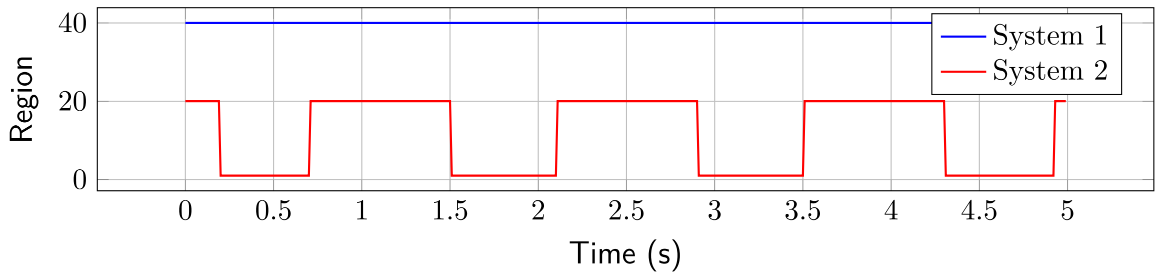


Figure 7-17: The traffic model regions the systems are in over time, corresponding to Figure 7-16

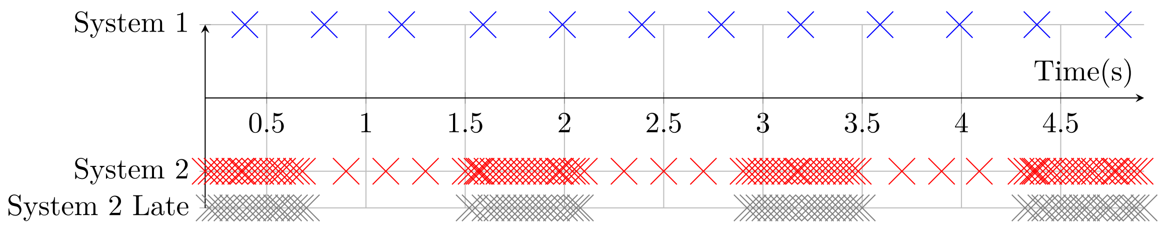


Figure 7-18: Triggering Times for both systems corresponding to Figure 7-16. Early triggers are denoted by a ‘X’, triggers on the deadline are denoted with a ‘O’, and late triggers for the second system are shown separately.

Chapter 8

Conclusions and Future Work

Several techniques and algorithms were introduced and implemented to safely schedule PETC systems, with the main goal of finding a scalable synthesization algorithm. This was achieved by abstracting the PETC systems by TSs and solving safety games on those. To increase efficiency, the TSs were reduced in size by grouping several states, as well as using BDDs to represent them and apply the operator symbolically.

Additionally, to increase schedulability, the subsystems are extended to occasionally allow late triggers. However, stability when using these late triggers has not been formally shown. This approach does show though that extension to these models can be quite easily designed and integrated into the system.

8-1 Future Work

There are multiple modifications and extensions, both theoretical and practical, that expand upon this thesis and fix some minor shortcomings that could be explored in the future:

- A practical implementation of a resulting scheduler, for example using a hardware-in-the-loop simulation with techniques from [21] and [5].
- The traffic models can be refined such that they incorporate also the future triggering behaviour [25, 26]. This can result in a more deterministic transition system (and more states) potentially increasing its schedulability. With a little modification to the construction of the Transition Systems (TSs), this technique could be incorporated if synthesization fails.
- A theoretical analysis of stability of late triggers could be performed, mainly to derive for which values of r and Δ the system would remain stable.
- The discussed techniques can in principle be used for any ETC system and abstraction algorithm as long as the resulting abstractions can be converted to TSs. One could for

example follow the approach from [27] with some modifications to create traffic models for general nonlinear PETC systems.

- The synthesization approach could be extended to a collection of subsystems which have different triggering times, as long as all of the trigger times are an integer multiple of some common value: $h_i = n_i \cdot a$, $n_i \in \mathbb{N}$. The TSs should be modified to add 'in-between'-states and a 'nothing'-action. However, this approach would add a lot of states.

- Since synthesizing a scheduler takes a lot of computation time, it would be nice to be able to determine whether a collection of subsystem is schedulable beforehand. Most likely the only way to know this with certainty is to perform the synthesization procedure. However, a set of heuristics (such as the Minimum inter-event time (MIET)) could be developed, which would indicate if system is likely to be unschedulable.

- In the partitioning algorithm, if synthesization fails, all the systems are refined. However, a collection of heuristics could be used to determine which subsystems to refine and how much, possible speeding up the process.

- Instead of adding 'late' states to all trigger states, one could only add these to bottleneck states. This would greatly decrease the number of states, however, this would require more manual labour or some heuristic to determine to which states to add.

- For some systems, the solution Z of the safety game could be only a small subset of the complete state-set. If such a system has an initial state outside of Z, it cannot find a safe input. A reachability game could be solved to reach this solution Z as fast as possible. However, this could possibly result in a collision of transmissions. A possible solution to resolve this issue is to add 'late' states only to the states outside the solution Z. This would decrease stability (or even cause instability) of the system, but possibly allows to reach Z from any initial condition.

- A more efficient refinement algorithm using Binary Decision Diagrams (BDDs).

- Modify the resulting schedulers (by removing some possible inputs for example) to maximize the average inter-event time.

- In this thesis, it is assumed that only the control loops are connected to a Networked Control Systems (NCS). However, in general networks, other unrelated systems are connected to the network as well. The additional network traffic that these nodes would cause could be modelled by one or more additional TSs and integrated into the complete system.

Appendix A

Simulation of Systems 7-6 and 7-7, with different triggering conditions

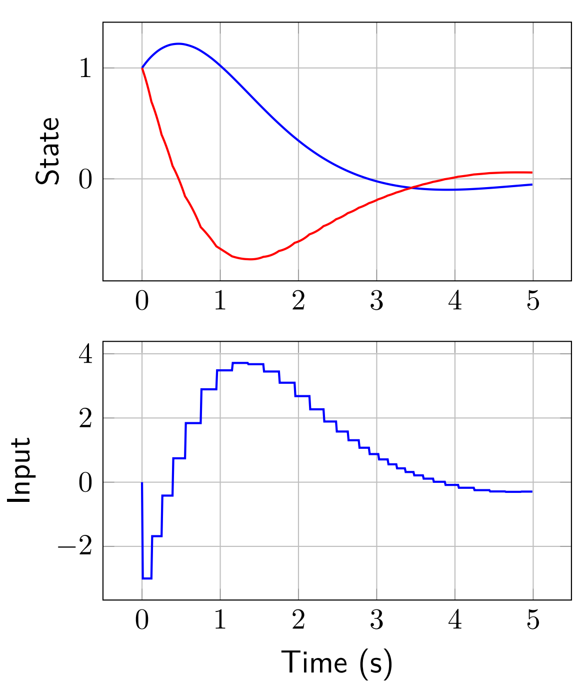
In [3], systems 7-6 and 7-7 have been successfully scheduled using a different triggering condition:

$$\Gamma(z) = \begin{bmatrix} \xi^T \\ \hat{\xi}^T \end{bmatrix}^T \begin{bmatrix} (1-\sigma^2)I & -I \\ -I & I \end{bmatrix} \begin{bmatrix} \xi \\ \hat{\xi} \end{bmatrix} = |e|^2 - \sigma\,|\xi|^2\,, \tag{A-1}$$

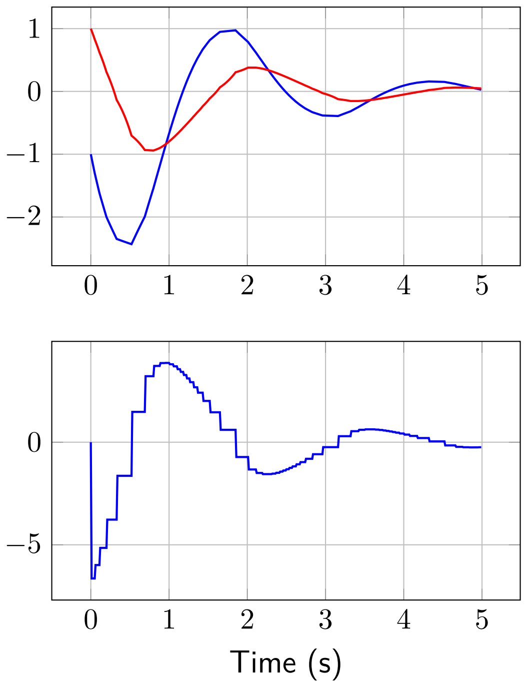
where $\sigma = 0.05$. The two systems share this same triggering condition. This results in abstractions with regions (with $h = 0.01$ and $k_{max} = 20$):

$$\begin{aligned} \mathcal{R}_1 &= \{\mathcal{Q}_{11}, \ldots, \mathcal{Q}_{20}\} \\ \mathcal{R}_2 &= \{\mathcal{P}_4, \ldots, \mathcal{P}_{20}\} \end{aligned} \tag{A-2}$$

For reference, a scheduler has also been synthesized using this triggering condition, for which the results are shown in Figures A-1, A-2 and A-3.



(a) Simulation of the plant 7-6 with initial condition $\xi_0 = [1, 1]^T$, $h = 0.01$ and $k_{max} = 20$ and triggering condition A-1.



(b) Simulation of the plant 7-7 with initial condition $\xi_0 = [1, -1]^T$, $h = 0.01$, $k_{max} = 20$ and triggering condition A-1.

Figure A-1

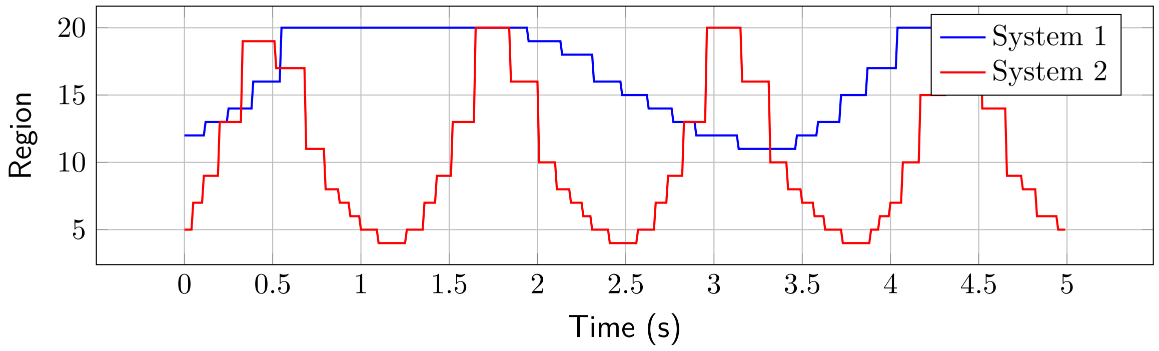


Figure A-2: The traffic model regions the systems are in over time, corresponding to Figure A-1

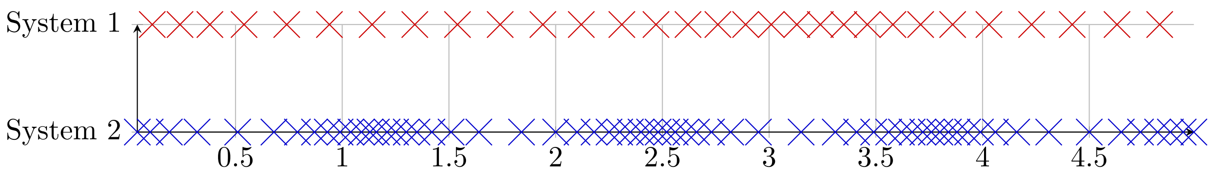


Figure A-3: Triggering Times for both systems corresponding to Figure A-1. Early triggers are denoted by a ‘X’, and triggers on the deadline are denoted with a ‘O’.

Appendix B

Proofs

B-1 Proof Proposition 3-3.1

In the following system S is created as is described in Section 3-1-1 and system S^I is the partitioned system as described in Section 3-3-1. Here it is shown that $S' \preceq_{AS} S$.

Proof. In the following, $\mathcal{T} = \{s \mid H(s) = T\}$. By construction, S^I is alternatingly simulated by $\bar{S}$: $S^I \preceq_{AS} \bar{S}$. This can be shown by defining the relation

$$R = \{(b, s) \mid s \in \bar{X},\, b := [s]_H\}, \tag{B-1}$$

where $[s]_Q$ is the equivalence class on $\bar{X}$ defined by the equivalence relation Q induced by the output map H:

$$Q := \{(x, y) \in \bar{X} \times \bar{X} \mid \bar{H}(x) = \bar{H}(y)\}. \tag{B-2}$$

The first condition of alternating simulation is satisfied by setting $\bar{X}_0 = \mathcal{T}$ and $X_0^I = \{T\}$. The second condition is satisfied by definition of the relation R. The final condition can be shown to be satisfied by considering four separate cases:

- $b = T_1$: The only state in $\bar{S}$ related to b is $s = T_1$. The only possible action in b is $U(b) = \{t\}$. The only possible actions in s is $U(s) = \{t\}$. Choose $u_{S^I} = t$: $Post_t(T_1) = \{T_1\}$. Then if $u_{\bar{S}} = t$: $Post_t(s) = \{T_1\}$ and $(T_1, T_1) \in R$.
- $b = T$:
  All states in $\bar{S}$ related by R to b are: $s \in \mathcal{T}$. The possible actions in b are $U(b) = \{t, w\}$. The possible actions in s are $U(s) = \{t, w\}$.
  - $u_{S^I} = t$: $Post_t(b) = \{b\}$. Then if $u_{\bar{S}} = t$: $Post_t(s) \subseteq \mathcal{T}$. By definition of R: $\forall y \in \mathcal{T} : (b, y) \in R$
  - $u_{S^I} = w$: $Post_t(b) \subseteq \{W_{k_{max}-1}, W_{k_{max}-2}, \ldots\}$. Then if $u_{\bar{S}} = w$: $Post_t(s) = \{W_{n,1}\}$ for some n dependent on s. By definition of R: $\forall y \in \mathcal{T} : (b, y) \in R$

- $b = W_n$, $n > 0$: All related states are of the form $s = W_{n+l,l}$. Additionally, $U(b) = \{t, w\}$ and $U(s) = \{t, w\}$.
  - $u_{S^I} = t$: $Post_t(b) = \{T\}$. Then if $u_{\bar{S}} = t$: $Post_t(s) \subseteq \mathcal{T}$. By definition of R: $\forall y \in \mathcal{T} : (b, y) \in R$
  - $u_{S^I} = w$: $Post_t(b) \subseteq \{W_{n-1}, \dots, W_0\}$. Then if $u_{\bar{S}} = w$: $Post_t(s) = \{W_{n+l+1,l}\}$. By definition of R: $(W_{n-1}, W_{n+l+1,l}) \in R$.
- $b = W_0$: All related states are of the form $s = W_{l,l}$. $U(b) = U(s) = \{t\}$. If $u_{S^I} = t$: $Post_t(b) = \{T\}$. Then if $u_{\bar{S}} = t$: $Post_t(s) \subseteq \mathcal{T}$. By definition of R: $\forall y \in \mathcal{T} : (b, y) \in R$.

Thus the third condition is also satisfied and $S^I \preceq_{AS} \bar{S}$ holds. □

B-2 Proof of Proposition 3-3.3

Proof. In this proof it is shown that $S^{(i+1)} \preceq_{AS} S^{(i)}$.
Since refinement is performed with every block $C \in X^{(i)}$, each block $b^{(i)} \in X^{(i)}$ is split apart at most $n := \left|X^{(i)}\right|$ times, i.e.

$$\begin{aligned} b^{(i)} \xrightarrow{split} \{&b^{(i)} \cap Pre(c_1^{(i)}) \cap Pre(c_2^{(i)}) \cap \dots \cap Pre(c_n^{(i)}), \\ &b^{(i)} \cap Pre(c_1^{(i)}) \cap Pre(c_2^{(i)}) \cap \dots \setminus Pre(c_n^{(i)}), \dots, \\ &b^{(i)} \setminus Pre(c_1^{(i)}) \setminus Pre(c_2^{(i)}) \setminus \dots \setminus Pre(c_n^{(i)})\} \\ =:\ & f^{(i)}(b^{(i)}). \end{aligned} \tag{B-3}$$

Then the relation R can be defined as:

$$R := \{(a, b^{(i)}) \in X^{(i+1)} \times X^{(i)} |\, a \in f^{(i)}(b^{(i)})\}. \tag{B-4}$$

Alternating simulation between the two systems will be shown using this relation, but first note that if $(b^{(i)}, u, c^{(i)}) \in \xrightarrow[S^{(i)}]{}$, then there is at least one transition $(a, u, a') \in \xrightarrow[S^{(i+1)}]{}$, where $a \in f^{(i)}(b^{(i)})$, $a' \in f^{(i)}(c^{(i)})$. This implies that,

$$Post_u(a) \subseteq \bigcup_{c^{(i)} \in Post_u(b^{(i)})} f^{(i)}(c^{(i)}), \tag{B-5}$$

where $b^{(i)})$ is the originating block of a, i.e. $a \in f^{(i)}(b^{(i)})$. Thus $\forall (a, b^{(i)}) \in R$ and $\forall u_a \in U^{(i+1)}(a)$, there exists $u_b \in U^{(i)}(b^{(i)})$ (namely $u_b = u_a = u$), such that $\forall c^{(i)} \in Post_u(b^{(i)})$ there exists $a' \in Post_u(a)$, namely the $a' \in f^{(i)}(c^{(i)})$ for which the transition (a, u, a') exists, such that $(a', c^{(i)}) \in R$.
By construction of $X_0^{(i+1)}$ and $H^{(i+1)}(x)$, the first two conditions of alternating simulation are also satisfied, thus $S^{(i+1)} \preceq_{AS} S^{(i)}$.

□

B-3 Proof of Proposition 3-3.5

Proof. After maximally refining a Transition System (TS), it is known that $S' \cong S$ (Section 2-1-2). Denote the corresponding bisimulation relation as R. Additionally, it is given that for some $W \subseteq X$ a solution to the safety game (Definition 2-1.9) for system S exists, with solution Z, i.e.:

$$Z = F_W(Z) = \{x \in Z | \, x \in W \, \wedge \, \exists u : \, \emptyset \neq Post_u(x) \subseteq Z\}. \tag{B-6}$$

The goal is to show that there also exists a solution to the safety game for the system S' with $W' := \{x' \in X' | \, \exists x \in W : \, (x', x) \in R\}$. $F_{W'}(Z')$ can be expanded as follows:

$$\begin{aligned} F_{W'}(Z') : &= \{x' \in Z' | \, x' \in W' \, \wedge \, \exists u' : \, \emptyset \neq Post'_{u'}(x') \subseteq Z'\} \\ &= \{x' \in Z' | \, \exists x \in X : \, (x', x) \in R \, \wedge \, x \in W \, \wedge \, \exists u : \, \emptyset \neq Post_u(x) \subseteq Z\} \\ &= \{x' \in Z' | \, \exists x \in X : \, (x', x) \in R \, \wedge \, x \in F_W(Z)\} \\ &= \{x' \in Z' | \, \exists x \in X : \, (x', x) \in R \, \wedge \, x \in Z\} \end{aligned} \tag{B-7}$$

Then choosing Z' as:

$$Z' := \{x' \in X' | \, \exists x \in X : \, (x', x) \in R \, \wedge \, x \in Z\}, \tag{B-8}$$

solves the safety game for S' with W' as $Z' = F_{W'}(Z')$.
Thus if the safety game for some system can be solved, a solution will be found for all bisimilar systems (with a suitable safety set). □

Bibliography

[1] L. Bushnell and H. Ye, *Networked Control Systems: Architecture and Stability Issues*, pp. 1–9. London: Springer London, 2013.

[2] J. Lunze and L. Grüne, *Introduction to Networked Control Systems*, pp. 1–30. Heidelberg: Springer International Publishing, 2014.

[3] D. Adzkiya and M. Mazo Jr., "Scheduling of Event-Triggered Networked Control Systems using Timed Game Automata," *arXiv e-prints*, p. arXiv:1610.03729, Oct. 2016.

[4] P. Schalkwijk, *Automating scheduler design for Networked Control Systems with Event-Based Control: An approach with Timed Automata.* M.s. thesis, 2019.

[5] A. A. Samant, *Scheduling Strategies for Event-Triggered Control Using Timed Game Automata Over CAN Networks.* M.s. thesis, 2020.

[6] G. de Albuquerque Gleizer and M. Mazo, "Scalable traffic models for scheduling of linear periodic event-triggered controllers," in *Proceedings IFAC World Congress 2020*, IFAC, 2020.

[7] P. Tabuada, *Verification and Control of Hybrid Systems: A Symbolic Approach.* Springer Science Business Media, 2009.

[8] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series).* The MIT Press, 2008.

[9] W. P. M. H. Heemels, K. H. Johansson, and P. Tabuada, "An introduction to event-triggered and self-triggered control," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 3270–3285, 2012.

[10] W. P. M. H. Heemels, M. C. F. Donkers, and A. R. Teel, "Periodic event-triggered control for linear systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 4, pp. 847–861, 2013.

[11] C. Fiter, L. Hetel, W. Perruquetti, and J.-P. Richard, "A state dependent sampling for linear state feedback," *Automatica*, vol. 48, no. 8, pp. 1860–1867, 2012.

[12] A. Fu and M. Mazo, "Traffic models of periodic event-triggered control systems," *IEEE Transactions on Automatic Control*, vol. 64, no. 8, pp. 3453–3460, 2019.

[13] C. Y. Lee, "Representation of switching circuits by binary-decision programs," *The Bell System Technical Journal*, vol. 38, no. 4, pp. 985–999, 1959.

[14] Akers, "Binary decision diagrams," *IEEE Transactions on Computers*, vol. C-27, no. 6, pp. 509–516, 1978.

[15] R. E. Bryant, "Symbolic boolean manipulation with ordered binary-decision diagrams," *ACM Comput. Surv.*, vol. 24, no. 3, p. 293–318, 1992.

[16] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. 35, no. 8, p. 677–691, 1986.

[17] A. Girard, "Dynamic triggering mechanisms for event-triggered control," *IEEE Transactions on Automatic Control*, vol. 60, pp. 1992–1997, July 2015.

[18] A. Szymanek, G. d. A. Gleizer, and M. Mazo, "Periodic event-triggered control with a relaxed triggering condition," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 1656–1661, 2019.

[19] C. Control and D. Systems, "dd (version 0.5.6.)." https://pypi.org/project/dd/, 2020.

[20] F. Somenzi, "Cudd: Cu decision diagram package release 2.2.0," 1998.

[21] A. R. Arjona, *Implementing Symbolic Controllers into FPGAs.* M.s. thesis, 2019.

[22] M. Donkers, *Networked and event-triggered control systems.* PhD thesis, Mechanical Engineering, 2011.

[23] P. Tabuada, "Event-triggered real-time scheduling of stabilizing control tasks," *IEEE Transactions on Automatic Control*, vol. 52, no. 9, pp. 1680–1685, 2007.

[24] L. Hetel, A. Kruszewski, W. Perruquetti, and J. Richard, "Discrete and intersample analysis of systems with aperiodic sampling," *IEEE Transactions on Automatic Control*, vol. 56, no. 7, pp. 1696–1701, 2011.

[25] G. d. A. Gleizer and M. Mazo, "Towards traffic bisimulation of linear periodic event-triggered controllers," *IEEE Control Systems Letters*, vol. 5, no. 1, pp. 25–30, 2021.

[26] G. de A. Gleizer and M. Mazo, *Computing the Sampling Performance of Event-Triggered Control.* New York, NY, USA: Association for Computing Machinery, 2021.

[27] G. Delimpaltadakis and J. Mazo, Manuel, "Traffic Abstractions of Nonlinear Homogeneous Event-Triggered Control Systems," *arXiv e-prints*, p. arXiv:2003.09361, Mar. 2020.

Glossary

List of Acronyms

NCS	Networked Control Systems
ETC	Event Triggered Control
STC	Self-Triggered Control
CETC	Continuous Event Triggered Control
PETC	Periodic Event Triggered Control
MIET	Minimum inter-event time
BDD	Binary Decision Diagram
OBDD	Ordered Binary Decision Diagram
ROBDD	Reduced Ordered Binary Decision Diagram
TS	Transition System