Master of Science Thesis

Automating scheduler design for Networked Control Systems with Event-Based Control

an approach with Timed Automata

P.C. Schalkwijk

June 21, 2019

Delft Center for Systems and Control

Delft University of Technology

Automating scheduler design for Networked Control Systems with Event-Based Control

an approach with Timed Automata

Master of Science Thesis

For obtaining the degree of Master of Science in Systems and Control
at Delft University of Technology

P.C. Schalkwijk

June 21, 2019

Faculty of Mechanical, Maritime and Materials Engineering · Delft University of Technology

Delft University Of Technology
Delft Center for Systems and Control

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering for acceptance a thesis entitled **"Automating scheduler design for Networked Control Systems with Event-Based Control"** by **P.C. Schalkwijk** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: June 21, 2019

Supervisor: ______________________
dr. M. Mazo Jr.

Readers: ______________________
ir. G. Gleizer

____
dr. ir. S.E. Verwer

Abstract

As the use of Networked Control Systems increases, the need for control methods with more efficient network usage also grows. These methods require a more sophisticated way of predicting their traffic, and an approach for this is using a formal modelling approach using Timed Automata. Timed Automata have been used for over 25 years for several scheduling problems, but have not been adopted by the control systems community for scheduling event-triggered systems. This is a recent development for which no easy to use software tools have been developed, and performance in real-world applications is yet untested.

In this master thesis, an existing approaches for scheduling event-triggered controllers is implemented in a set of tools. This approach creates abstractions of communication traffic, models them as timed automata and finds a scheduler avoiding communication conflicts. This set of tools is used to test the scalability with respect to abstraction accuracy and number of systems that can be connected. The set of tools can be used in the future to further improve on the techniques used.

Acknowledgements

I would like to thank my supervisor dr. M. Mazo Jr. for providing me with an interesting thesis subject, that matched my skills and interests. Also, I want to thank him for his supervision during entire process of my thesis project, and for organising weekly meetings with my peers. During these meetings I have learned a lot, both relevant and irrelevant for my thesis. I would like to thank ir. G. Gleizer for his help getting me started with programming traffic abstractions. I would like dr. ir. S.E. Verwer for being on my committee and taking the time to take an in-depth look into my research. Finally I would like to thank the Delft University of Technology for being a constant factor in my life for the past decade.

Delft, University of Technology
June 21, 2019

P.C. Schalkwijk

Table of Contents

List of Figures

Chapter 1

Introduction

The growing scale and complexity of control systems, and the current advancements in enabling technologies impose the next step towards Networked Control Systems (NCS). In NCS controller, sensors and actuators can be spatially distributed and communicate digitally over a band-limited network layer, an example shown in Figure 1-1 Advantages of using NCS are for example increased flexibility, scalability and reconfigurability.

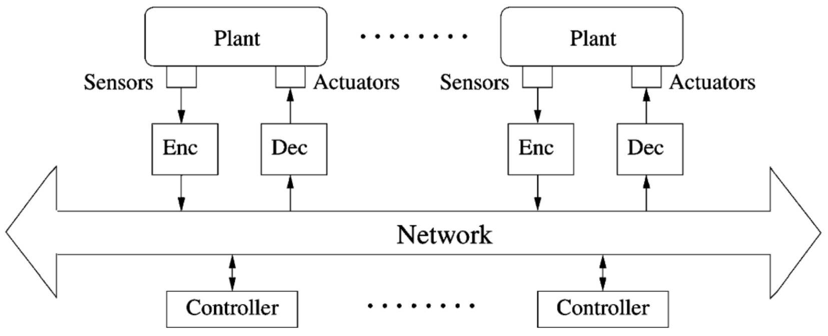


Figure 1-1: A schematic overview of a networked control system [1]

Applications of NCS can be found in a wide range of fields. Apart from its apparent advantages, there are drawbacks to using networked control systems. We focus on one in particular: resource constraints of the network channel. Event-based methods [9, 10, 11, 12] can minimize the resource usage of a control-loop, and allow more control loops to be used on the same network. These methods explicitly address constraints regarding energy, computation and communication.

In Event-triggered Control (ETC), a triggering condition based on current measurements is continuously monitored. On violation of the condition, an event is triggered and the control input is updated. Usually, the actuator remembers the last control input in a sample-and-hold manner.

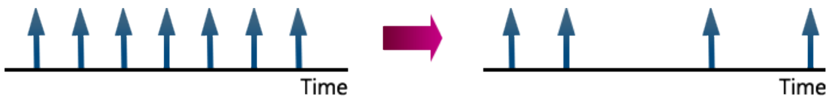


Figure 1-2: Events from time-triggered (left) to event-triggered control (right) [2]

Event-based control approaches are no longer guaranteed to have a periodic update. As schematically shown in Figure 1-2, the time between events is no longer constant. This aperiodic behaviour creates a new challenge: the scheduling of transmissions. The maximum number of messages that can be transmitted over the network at the same time, is limited by the finite number of channels present in the communication network. If the network has a number of control loops up to the number of channels, no scheduling is necessary. If there are more control loops than channels, as found in practice, a scheduler is necessary to determine which control loop has access to the network, while guaranteeing the stability of all control loops. The general architecture for a networked embedded controller is shown in Figure 1-3.

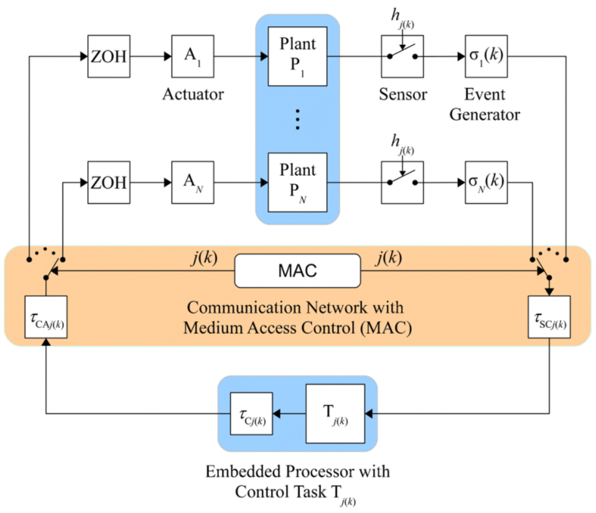


Figure 1-3: General architecture of a networked embedded control system with medium access constraints [3]

Several approaches to scheduler design exist in literature, ranging from static schedules determined before runtime [13, 14] to state dependent scheduling [15, 16, 17]. In more advanced approaches, joint design methods [3, 18] take both resource distribution and resource utilisation into account by co-designing control law, scheduler and event generator.

In [5], a different approach for scheduler design is proposed. This approach builds on the idea presented in [4], where the inter-event time of a event-triggered system is characterized by abstracting the state-space into regions. For each region in the state-space, the inter-event time is lower- and upper bounded, illustrated in Figure 1-4. It is shown that this characterization can be formally defined as a timed automaton, shown schematically in Figure 1-5. In [5], multiple timed automata are connected in a network of timed automata, for which a scheduler is designed using reachability analysis. This scheduler either waits for an event to trigger naturally, forces controller updates at an earlier time. To prevent an undesired over-use of forced controller updates, the number of consecutive updates is limited.

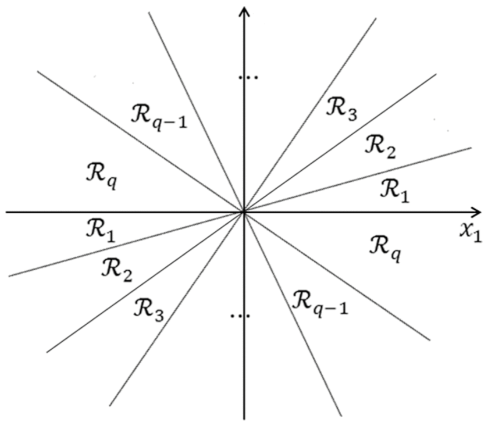
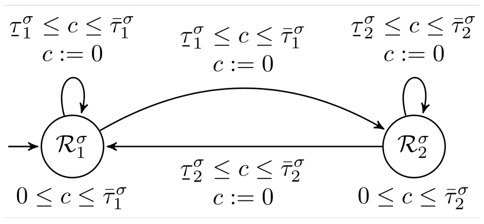


Figure 1-4: Illustrative example of the state-space abstraction using convex polyhedral cones [4]

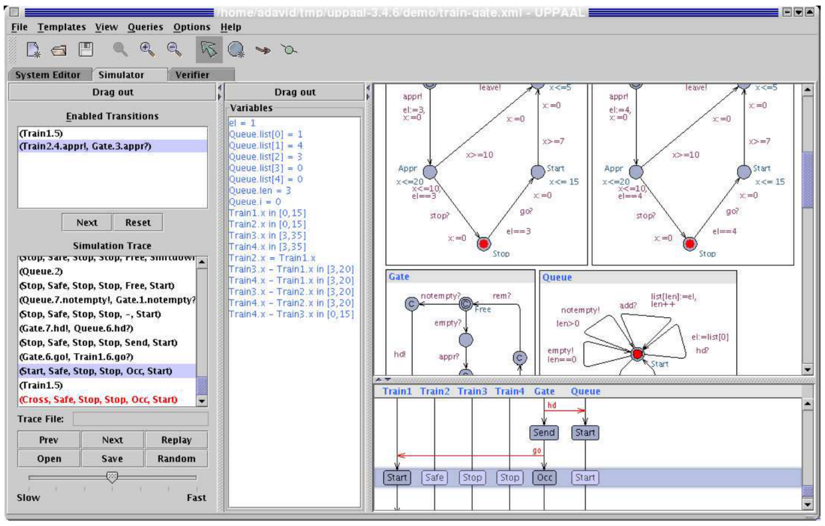


$\underline{\tau}_1^\sigma \le c \le \bar{\tau}_1^\sigma$ $c := 0$
$\underline{\tau}_1^\sigma \le c \le \bar{\tau}_1^\sigma$ $c := 0$
$\underline{\tau}_2^\sigma \le c \le \bar{\tau}_2^\sigma$ $c := 0$
$\mathcal{R}_1^\sigma$
$\mathcal{R}_2^\sigma$
$\underline{\tau}_2^\sigma \le c \le \bar{\tau}_2^\sigma$ $c := 0$
$0 \le c \le \bar{\tau}_1^\sigma$
$0 \le c \le \bar{\tau}_2^\sigma$

Figure 1-5: Control loop modelled as a Timed Automaton [5]

Timed Automata (TA) [19] are used as a framework to capture both qualitative and quantitative features of real-time systems with finite state machines extended with clocks. Timed Game Automata (TGA), extending TA with game theory, can model a game of controller versus environment, modelling uncertainty. By extending TGA with pricing, systems can be analysed with respect to some defined cost. Uppaal [6] can verify real-time systems described with timed automata, and is still actively maintained, whereas other tools such as HyTech [20] and the TIMES tool [21] have not been updated for several years. Uppaals graphical user interface is shown in Figure 1-6

Uppaal has several extensions, using TA for calculation of optimal paths, timed games and stochastic behaviour using both formal and statistical analysis methods. The limiting factor when modelling with timed automata is the number of clocks in the model: the number of states increases exponentially with the number of clocks [19].



/home/adavid/tmp/uppaal-3.4.6/demo/train-gate.xml - UPPAAL
File Templates View Queries Options Help
System Editor Simulator Verifier
Drag out
Enabled Transitions
Next Reset
Simulation Trace
Trace File:
Prev Next Replay
Open Save Random
Slow Fast
Drag out
Variables
Gate
Queue
Train1 Train2 Train3 Train4 Gate Queue
Send Start
Start Safe Stop Stop Occ Start

```
(Train1.5)
(Train2.4.appr!, Gate.3.appr?)
```

```
(Stop, Safe, Stop, Stop, Free, Shutdown
(Queue.2)
(Stop, Safe, Stop, Stop, Free, Start)
(Queue.7.notempty!, Gate.1.notempty?
(Stop, Safe, Stop, Stop, -, Start)
(Gate.7.hd!, Queue.6.hd?)
(Stop, Safe, Stop, Stop, Send, Start)
(Gate.6.go!, Train1.6.go?)
(Start, Safe, Stop, Stop, Occ, Start)
(Train1.5)
(Cross, Safe, Stop, Stop, Occ, Start)
```

```
el = 1
Queue.list[0] = 1
Queue.list[1] = 4
Queue.list[2] = 3
Queue.list[3] = 0
Queue.list[4] = 0
Queue.len = 3
Queue.i = 0
Train1.x in [0,15]
Train2.x in [0,15]
Train3.x in [3,35]
Train4.x in [3,35]
Train2.x = Train1.x
Train3.x - Train1.x in [3,20]
Train4.x - Train1.x in [3,20]
Train3.x - Train2.x in [3,20]
Train4.x - Train2.x in [3,20]
Train4.x - Train3.x in [0,15]
```

Figure 1-6: Illustrative example of a scheduling sequence modelled as a timed automaton in Uppaal [6]

For simulating real-time continuous system dynamics and network behaviour, one of the tools available is TrueTime, which extends MATLAB Simulink with existing network implementations and allows for extensions into other network implementations.

The goal of this work is to ease the use of the approach for scheduler design in [5] using Traffic Abstractions and Timed Automata for Scheduler design by creating a tool set that can easily be used to do system modelling, and that can be extended in future work to include other types of abstractions.

First, we will set the preliminaries by expanding on the traffic abstraction in chapter 2, by detailing modelling in chapter 3 and actual scheduler design in chapter 4.

Next, the problem statement in chapter 5 defines the design criteria for the tool set we want to design, and a useful test case is proposed. In chapter 6, the software implementation of the tool set is explained, linking the software to their theoretical counterparts in earlier chapters. In chapter 7, the test case relating to scalability is explained in more detail, and it's results are discussed.

Finally, in chapter 8 this work is concluded and ideas and recommendations for future work are proposed.

Chapter 2

Abstractions

In this work we use an abstracted system capturing the sampling behaviour of a family of event-triggered control systems, derived using the formal approach for as proposed in [4] and [5]. In this chapter we will first summarize the abstraction method by revisiting the state-space partitioning, calculating the upper and lower bounds and the use of reachability analysis to find transitions of the system.

2-1 Event Triggered Control Systems

Following [4, 5], we will consider the traffic of Linear Time-Invariant (LTI) systems of the form

$$\dot{\xi} = A\xi(t) + Bv(t),\ \ \xi(t) \in \mathbb{R}^n,\ \ v(t) \in \mathbb{R}^m \tag{2-1}$$

with linear state-feedback law implemented using a sample-and-hold manner:

$$v(t) = v(t_k) = K\xi(t_k),\ \forall t \in [t_k, t_{k+1}[,\ k \in \mathbb{N}_0 \tag{2-2}$$

We will use the sampling triggering law proposed in [11], also used in both [4, 5]:

$$t_{k+1} = min\{t \mid t > t_k \text{ and } |e(t)|^2 \geq \alpha|\xi(t)|^2\}, \alpha \in (0, \bar{\sigma}) \subset \mathbb{R}^+ \tag{2-3}$$

with σ the triggering coefficient, and measurement error $e(t)$:

$$e(t) = \xi(t_k) - \xi(t), t \in [t_k, t_{k+1}), k \in \mathbb{N}_0 \tag{2-4}$$

As in [4, 5, 7], we denote a state sample at t_k by $\xi(t_k) = x$. The inter-sample time of the state x, $\tau_\sigma(x)$, is defined as the time between consecutive updates of the sampled state:

$$\tau_\sigma(x) = min\{t| \ |e(t)|^2 \geq \sigma|\xi(t)|^2 \text{ and } \xi(0) = x\} \tag{2-5}$$

Within the current sampling interval $[t_k, t_{k+1}]$ the evolution of the state ξ_x and measurement error e_x are defined as:

$$\xi_x(t_k + \sigma = \Lambda(\sigma)x) \tag{2-6}$$

$$e_x(t_k + \sigma) = [I - \Lambda(\sigma)]x \tag{2-7}$$

with $\sigma \in [0, t_{k+1} - t_k]$ and

$$\Lambda(\sigma) = [I + \int_0^\sigma e^{Ar} dr (A + BK)] \tag{2-8}$$

Now we combine the sampled triggering law (2-5) with the state (2-6) and error (2-7) evolution in the current sampling interval to express the inter-sample time $\tau(x)$ as:

$$\tau(x) = \min\{\sigma > 0 | x^T \Phi(\sigma) x = 0\} \tag{2-9}$$

where $\Phi(\sigma)$ is defined:

$$\Phi(\sigma) = [I - \Lambda(\sigma)^T][I - \Lambda(\sigma)] - \alpha \Lambda^T(\sigma)\Lambda(\sigma) \tag{2-10}$$

2-2 Partitioning the state-space

To remove spatial dependency In the work of [4], the state-space is abstracted into a finite number of convex polyhedral cones $\mathcal{R}_s$ using *isotropic covering* as proposed in [22], where $s \in \{1, \dots, q\}$ and $\bigcup_{s=1}^{q} \mathcal{R}_s = \mathbb{R}^n$. General spherical coordinates are used such that $x \in \mathbb{R}^n : (r, \theta_1, \dots, \theta_{n-1})$, where $r = |x|$, $(\theta_1, \dots, \theta_{n-1})$ are the angular coordinates of x. We consider $(\theta_1, \dots, \theta_{n-2}) \in [0, \pi]$ and $\theta_{n-1} \in [-\pi, \pi]$. The angular coordinates are divided into $\bar{m}$ intervals, creating a total of $q = \bar{m}^{(n-1)}$ intervals.

Remark 2-2.1 *From [22].*
Excluding the origin, all the states which lie on a line that goes through the origin have the same inter-sample time, i.e., $\tau(x) = \tau(\lambda x), \forall \lambda \neq 0$.

Base on (2-2.1) only half the state space needs to be considered. This can for example be done by taking $\theta_{n-1} \in [0, \pi]$. Now, $q = 2 \times \bar{m}^{(n-1)}$. It is also by the notion of 2-2.1) that maps the infinite state-space to a finite abstraction, as a cone is the union of an infinite number of rays.
For two dimensional systems, the regions can be represented as:

$$\mathcal{R}_s = \{x \in \mathbb{R}^2 | x^T Q_s x \geq 0\}, \text{ if } n = 2 \tag{2-11}$$

for $s \in \{1, \dots, q\}$ and an appropriately designed $Q_s = Q_s^T \in \mathcal{M}_2(\mathbb{R})$. A graphical representation is taken from [7] and shown in Figure 2-1

For higher dimensional systems where $x \in \mathbb{R}^n | n \geq 3)$, following the work of [7], we define a region $\mathcal{R}_s$ by looking at its projections onto the $(n-1)$ two dimensional (x_i, x_{i+1})-planes. On each plane (x_i, x_{i+1}), the projection of $\mathcal{R}_s$ can be described with a matrix $Q_s^{i,i+1}$ in a similar manner as in **??**. The entire region can be described as the combination of these projections:

$$\mathcal{R}_s = \{x \in \mathbb{R}^n | x_{1,2}^T Q_s^{1,2} x_{1,2} \geq 0 \wedge x_{2,3}^T Q_s^{2,3} x_{2,3} \geq 0 \wedge \dots \wedge x_{n1,n}^T Q_s^{n1,n} x_{n1,n} \geq 0\}, \text{ if } n \geq 3. \tag{2-12}$$

For a system with $n = 3$, these projections can visualised as in In Appendix A the construction of these region matrices for 2D and nD systems is shown.

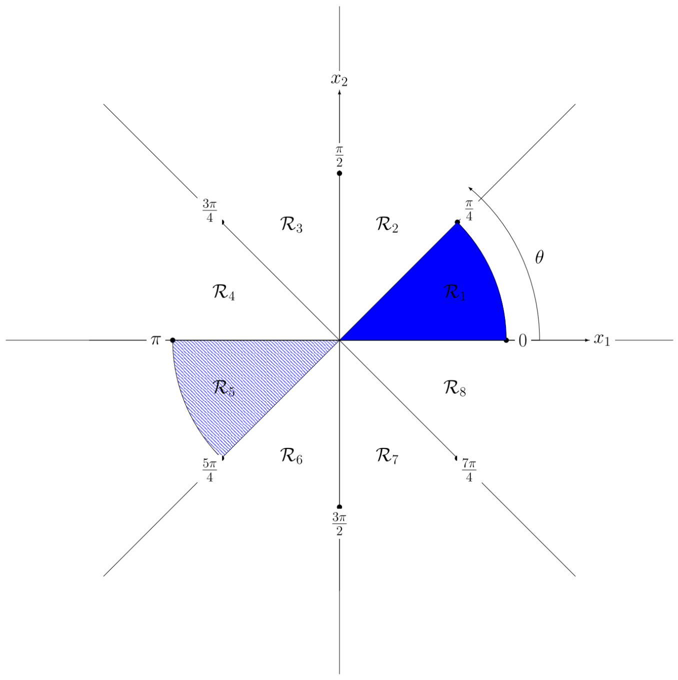


Figure 2-1: Partitioning of a two-dimensional state space with m = 4. Note that the blue arcs are only highlighting the regions R1 and R5 which have the same matrix Qs defining them, but are not bounding these regions. [7]

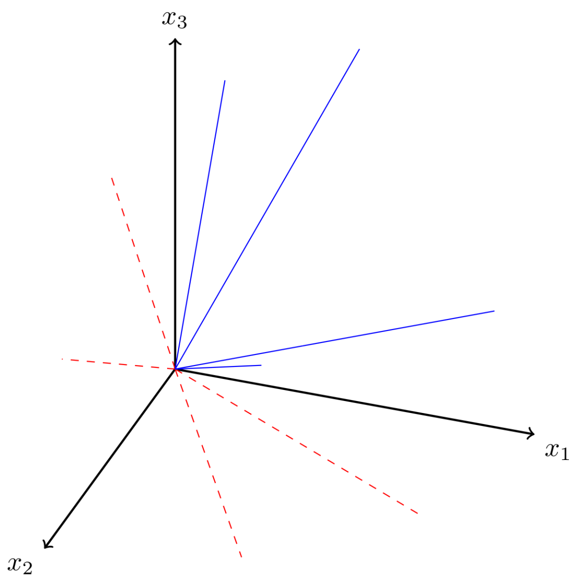


Figure 2-2: Example of a three-dimensional conic region (blue) and its projections onto the (x1, x2)-plane and the (x2, x3)-plane (red) [7]

2-3 Bounds on inter-event time

For each of the bounds on inter-sampling time a finite set of matrices $\Phi_{\kappa,s}$ is constructed.

2-3-1 Lower bound

For the lower bound $\underline{\tau}_s$, we construct $\underline{\Phi}_{\kappa,s}$ with $\kappa \in \mathcal{K}_s$:

$$(x^T \underline{\Phi}_{\kappa,s} x \leq 0, \forall \kappa \in \mathcal{K}_s) \implies (x^T \Phi(\sigma) x \leq 0, \forall \sigma \in [0, \underline{\tau}_s]) \tag{2-13}$$

We use the following lemma to construct $\underline{\Phi}_{\kappa,s}$:

Lemma 2-3.1 *From [4]. Consider a time limit $\underline{\tau}_s \in (0, \bar{\sigma}]$. If*

$$x^T \underline{\Phi}_{(i,j),s} x \leq 0 \qquad \forall (i,j) \in \mathcal{K}_s = (\{0, \dots, N_{conv}\} \times \{0, \dots, \frac{\underline{\tau} l}{\bar{\sigma}}\}) \tag{2-14}$$

then

$$x^T \Phi(\sigma) x \leq 0, \forall \sigma \in [0, \underline{\tau}_s]$$

with Φ as in Equation 2-10 and

$$\underline{\Phi}_{(i,j),s} := \hat{\underline{\Phi}}_{(i,j),s} + \underline{\nu} I \tag{2-15}$$

Where

$$\hat{\underline{\Phi}}_{(i,j),s} := \begin{cases} \sum_{k=0}^{i} L_{k,j} (\frac{\bar{\sigma}}{l})^k & \text{if } j < [\frac{\underline{\tau}_s l}{\bar{\sigma}}], \\ \sum_{k=0}^{i} L_{k,j} (\underline{\tau}_s - \frac{\bar{\sigma}}{l})^k & \text{if } j = [\frac{\underline{\tau}_s l}{\bar{\sigma}}], \end{cases} \tag{2-16}$$

$$\begin{cases} L_{0,j} := & I - \Pi_{1,j} - \Pi_{1,j}^T + (1-\alpha)\Pi_{1,j}^T \Pi_{1,j}, \\ L_{1,j} := & [(1-\alpha)\Pi_{1,j}^T - I]\Pi_{2,j} + \Pi_{2,j}^T [(1-\alpha)\Pi_{1,j} - I], \\ L_{k\geq 2,j} := & [(1-\alpha)\Pi_{1,j}^T - I]\frac{A^{k-1}}{k!}\Pi_{2,j} + \Pi_{2,j}^T \frac{(A^{k-1})^T}{k!}[(1-\alpha)\Pi_{1,j} - I] \\ & +(1-\alpha)\Pi_{2,j}^T (\sum_{i=1}^{k-1} \frac{(A^{i-1})^T}{i!} \frac{A^{k-i-1}}{(k-i)!})\Pi_{2,j} \end{cases} \tag{2-17}$$

$$\begin{cases} \Pi_{1,j} := & I + M_j(A + BK) \\ \Pi_{2,j} := & N_j(A + BK) \end{cases} \tag{2-18}$$

$$M_j := \int_0^{j\frac{\bar{\sigma}}{l}} e^{As} ds, \qquad N_j := AM_j + I \tag{2-19}$$

and

$$\underline{\nu} \geq \max_{\sigma' \in [0, \frac{\bar{\sigma}}{l}], r \in \{0, \dots, l-1\}} \lambda_{max}(\Phi(\sigma' + r\frac{\bar{\sigma}}{l}) - \tilde{\Phi}_{N_{conv},r}(\sigma')) \tag{2-20}$$

where

$$\tilde{\Phi}_{N_{conv},r}(\sigma')) := \sum_{k=0}^{N_{conv}} L_{k,r} \sigma'^k. \tag{2-21}$$

Proof. See [4]

The following approach regionally reduces the conservatism involved in Lemma 2-3.1 above using Linear Matrix Inequalities (LMI): the Regional Lower Bound Approximation as found in [4] for $n = 2$:

Theorem 2-3.1 (Regional Lower Bound Approximation[4]) *Consider the inter-sampling time set $\{\underline{\tau}_1, \ldots, \underline{\tau}_q\}$ and matrices $\underline{\Phi}_{\kappa,s}$ satisfying $\forall s \in \{1, \ldots, q\}, \forall \kappa = (i,j) \in \mathcal{K}_s$, $0 < \underline{\tau}_s \leq \bar{\sigma}$, $\Phi_{\kappa,s} \preceq 0$. If there exist scalars $\underline{\varepsilon}_{\kappa,s} \geq 0$ such that the following LMI*

$$\underline{\Phi}_{\kappa,s} + \underline{\varepsilon}_{\kappa,s} Q_s \preceq 0 \tag{2-22}$$

holds, then $\forall x \in \mathcal{R}_s$ as defined in Equation 2-11 the inter-sample time Equation 2-9 is lower bounded by $\underline{\tau}_s$.

As shown in [7], for higher dimensional systems ($n > 2$) where R_s is defined by Equation 2-12, the LMIs to consider are:

$$\underline{\Phi}_{\kappa,s} + \underline{\varepsilon}^{1,2}_{\kappa,s} \tilde{Q}^{1,2}_s + \underline{\varepsilon}^{2,3}_{\kappa,s} \tilde{Q}^{2,3}_s + \ldots + \underline{\varepsilon}^{n-1,n}_{\kappa,s} \tilde{Q}^{n-1,n}_s \preceq 0 \tag{2-23}$$

where $\underline{\varepsilon}^{i-1,i}_{\kappa,s}$ are nonnegative scalars.

2-3-2 Upper bound

Similarly, for the upper bound we construct $\bar{\Phi}_{\kappa,s}$ such that

$$(x^T \bar{\Phi}_{\kappa,s} x \geq 0, \forall \kappa \in \mathcal{K}_s) \implies (x^T \Phi(\sigma) x \geq 0, \forall \sigma \in [\bar{\tau}_s, \bar{\sigma}]) \tag{2-24}$$

following Lemma 2-3.2

Lemma 2-3.2 *From [4]. Consider a time limit $\bar{\tau}_s \in (\underline{\tau}_s, \bar{\sigma}]$. If*

$$x^T \bar{\Phi}_{(i,j),s} x \geq 0 \qquad \forall (i,j) \in \mathcal{K}_s = (\{0, \ldots, N_{conv}\} \times \{\frac{\bar{\tau} l}{\bar{\sigma}}, \ldots, l-1\}) \tag{2-25}$$

Then

$$x^T \Phi(\sigma) x \geq 0, \forall \sigma \in [\bar{\tau}_s, \bar{\sigma}]$$

with Φ as in Equation 2-10 and

$$\bar{\Phi}_{(i,j),s} := \hat{\bar{\Phi}}_{(i,j),s} + \bar{\nu} I \tag{2-26}$$

$$\hat{\bar{\Phi}}_{(i,j),s} := \begin{cases} \sum_{k=0}^{i} L_{k,j}((j+1)\frac{\bar{\sigma}}{l} - \bar{\tau}_s)^k & \text{if } j = [\frac{\bar{\tau}_s l}{\bar{\sigma}}], \\ \sum_{k=0}^{i} L_{k,j}(\frac{\bar{\sigma}}{l})^k & \text{if } j > [\frac{\bar{\tau}_s l}{\bar{\sigma}}] \end{cases}, \tag{2-27}$$

$$\bar{\nu} \geq \max_{\sigma' \in [0, \frac{\bar{\sigma}}{l}], r \in \{0, \ldots, l-1\}} \lambda_{max}(\Phi(\sigma' + r\frac{\bar{\sigma}}{l}) - \tilde{\Phi}_{N_{conv},r}(\sigma')) \tag{2-28}$$

and $L_{k,j}$ given by Equation 2-17, $\tilde{\Phi}$ given by Equation 2-21.
Proof. See [4]

Time instance $\bar{\sigma}$ is considered larger than the inter-sample time for any state in the state space:

$$x^T\Phi(\bar{\sigma})x \geq 0, \forall x \in \mathbb{R}^n \tag{2-29}$$

For the upper bounds, the conservatism per is regionally reduced Regional Upper Bound Approximation is defined in [4, 7]: for $n = 2$:

Theorem 2-3.2 (Regional Upper Bound Approximation[4]) *Consider the inter-sampling time set $\{bar\tau_1, \ldots, bar\tau_q\}$ and matrices $bar\Phi_{\kappa,s}$ satisfying $\forall s \in \{1, \ldots, q\}, \forall \kappa = (i,j) \in \mathcal{K}_s$, $\underline{\tau}_s < bar\tau_s \leq \bar{\sigma}$, $\Phi_{\kappa,s} \succeq 0$. If there exist scalars $\underline{\varepsilon}_{\kappa,s} \geq 0$ such that the following LMI*

$$\bar{\Phi}_{\kappa,s} + \bar{\varepsilon}_{\kappa,s}Q_s \succeq 0 \tag{2-30}$$

holds, then $\forall x \in \mathcal{R}_s$ as defined in Equation 2-11 the inter-sample time Equation 2-9 is lower bounded by $\underline{\tau}_s$.

As shown in [7], for higher dimensional systems ($n > 2$) where R_s is defined by Equation 2-12, the LMIs to consider are:

$$\bar{\Phi}_{\kappa,s} - \bar{\varepsilon}^{1,2}_{\kappa,s}\tilde{Q}^{1,2}_s - \bar{\varepsilon}^{2,3}_{\kappa,s}\tilde{Q}^{2,3}_s - \ldots - \bar{\varepsilon}^{n-1,n}_{\kappa,s}\tilde{Q}^{n-1,n}_s \succeq 0 \tag{2-31}$$

2-4 Reachability Analysis

To construct to transition map, in [5, 7, 4] reachability analysis is used together with the notion of *flow pipe*:

Definition 2-4.1 (Flow Pipe [5]) *The set of reachable states or the flow pipe ate the time interval $[t_1, t_2]$ from a set of initial states X_0 is denoted by*

$$\mathcal{X}_{[t_1,t_2]}(X_0) = \bigcup_{t\in[t_1,t_2]} \{\xi(t)|\xi(0) \in X_0\} \tag{2-32}$$

If the flow pipe $\mathcal{X}_{[t_1,t_2]}(X_0)$ intersects with a region $\mathcal{R}_s$ starting from an initial set X_0, a transition from the region containing X_0 to $\mathcal{R}_s$. A visualisation for a flow pipe is shown in Figure 2-3

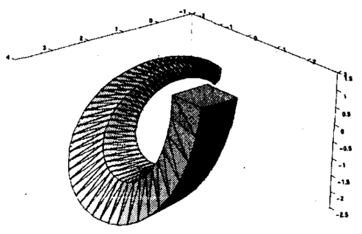


Figure 2-3: A flow pipe computed for a 3 dimensional system [8]

Chapter 3

Modelling

In the first section of this chapter we show the abstractions from chapter 2 will be modelled using Timed Game Automata (TGA), as done in [5]. In the final section, we show how such a TGA is modelled in Uppaal.

A Timed Automata (TA)) is a finite state automaton with a set of non negative real-valued variables referred to as 'clocks'. These real-valued variables model logical clocks, and are incremented simultaneously at the same rate, but can be reset individually. Clock constrains are used to restrict the behaviour of the automaton.

A clock constraint in a location is referred to as a *invariant*. Time is allowed to progress in a location while the invariant condition holds, and if the condition no longer holds, the location must be left.

A clock constraint on a transition is referred to as a *guard*. A transition over an edge is only allowed if all of its guards hold.

On their original introduction, TA used Büchi and Muller accepting conditions to enforce progress conditions [19]. The simplified version used here are referred to as Timed Safety Automata as described in [23]. These Timed Safety Automata use local invariant conditions to enforce progress conditions. We focus on Timed Safety Automata in this work, and refer to them as simply as TA, using the same notation as found in [5].

3-1 Timed Game Automata

For TGA we also follow the definitions in [5]: A TGA is a TA in which the set of actions is split into controllable and uncontrollable actions. The player, or scheduler in our case, can trigger controllable actions, and the opponent/environment can trigger uncontrollable actions. An example of a TGA is shown in Figure 3-1, with straight and dashed showing edges with controllable and uncontrollable actions respectively.

We define $\mathcal{C}$ as a set of finitely many clocks, $\texttt{Act}$ as the set of finitely many actions and $\mathbb{N}_0$ as the set of natural numbers including 0. A clock constraint, to be used as a guard or location

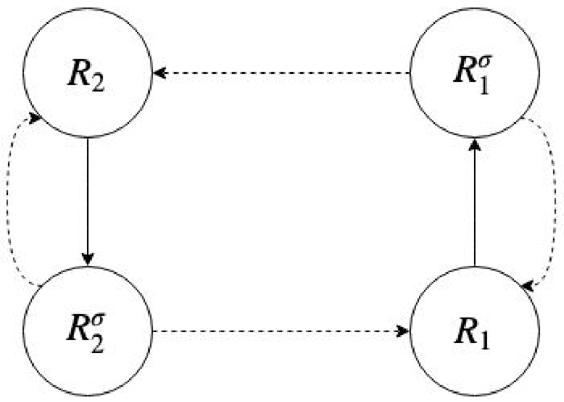


Figure 3-1: A TGA modelling a single Control Loop

invariant, has the form $x \bowtie n$ or $x - y \bowtie n$ for $x, y \in C$, $\bowtie \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}_0$. $\mathcal{B}$ denotes the set of clock constraints.

Definition 3-1.1 (Timed Automaton [5])

A timed automaton is a tuple $(L, l_0, \mathtt{Act}, C, E, \mathtt{Inv})$ *where*

- L *is a set of finitely many locations*
- l_0 *is the initial location*
- $\mathtt{Act}$ *is a set of finitely many actions*
- C *is a set of finitely many real-valued clocks*
- $E \subseteq L \times \mathtt{Act} \times B(C) \times 2^C \times L$ *is the set of edges*
- $\mathtt{Inv} :\rightarrow \mathcal{B}(C)$ *assigns invariants to locations*

Location invariants are restricted to constraints that are downwards closed, in the form: $c \leq n$ *or* $c < n$ *where* c *is a clock and* $n \in \mathbf{N}_0$

Definition 3-1.2 (Timed Game Automaton [5])

A timed game automaton is a tuple $(L, l_0, \mathtt{Act}_c, \mathtt{Act}_u, C, E, \mathtt{Inv})$ *where* $(L, l_0, \mathtt{Act}, C, E, \mathtt{Inv})$ *is a timed automaton and*

- $\mathtt{Act}_c$ *is a set of controllable actions*
- $\mathtt{Act}_u$ *is a set of uncontrollable actions*
- $\mathtt{Act} = \mathtt{Act}_c \cup \mathtt{Act}_u$
- $\mathtt{Act}_c \cap \mathtt{Act}_u = \emptyset$

Following [5], an edge from l to l' is described as $l \xrightarrow{g,a,\mathbf{r}} l'$, if $l, g, a, \mathbf{r}, l' \in E$ and as $l \to l'$ for an arbitrary label. The semantics of a TA (and TGA) are defined as a transition system where the current location and the current clock values are considered the state of the system. A transition can either be a delay of some time, or taking an enabled edge. Clock assignment functions $u : C \to \mathbb{R}_{\geq 0}$ are used to keep track of clock values, and $u \models g$ denotes that the clock values of u satisfy guard g. For $d \in \mathbb{R}_{\geq 0}$, $u + d$ denotes the clock assignment that maps all $c \in C$ to $u(c) + d$. For a set clocks $\mathbf{c} \subseteq C$, $u[\mathbf{c}]$ denotes the clock assignment that maps all clocks in $\mathbf{c}$ to 0, and agress with u for the rest of the clocks in $C \setminus \mathbf{c}$.

Definition 3-1.3 (Operational Semantics [5])

The semantics of a Timed (Game) Automaton is a transition system in which states are pairs of location l and clock assignment u, and transitions are defined by the rules:

- *Delay transition:* $(l, u) \xrightarrow[TS]{d} (l, u + d)$ *if* $u \models \texttt{Inv}(l)$ *and* $(u + d) \models \texttt{Inv}(l)$ *for a non-negative real number* $d \in \mathbb{R}_{\geq 0}$
- *Discrete transition:* $l, u) \xrightarrow[TS]{a} (l', u')$ *if* $l \xrightarrow{g,a,r} l'$, $u \models g$, $u' = u[\mathbf{r}]$ *and* $u' \models \texttt{Inv}(l'$

A run of a timed automaton is a sequence of alternating delay and discrete transitions in the transition system.

As in [5] we use $Runs(TGA)$ to denote the set of runs of a timed game automaton TGA starting from the initial state (l_0, u_0) where u_0 is a clock assignment that maps all $c \in C$ to 0. For a finite run ρ, the last state of the run is denoted by $last(\rho)$. The set of actions $\texttt{Act}$ (3-1.2) is assumed to consist of symbols for input actions $a?$, output actions $a!$ and internal actions $*$. Synchronous communication between different TA is done with handshake synchronisation

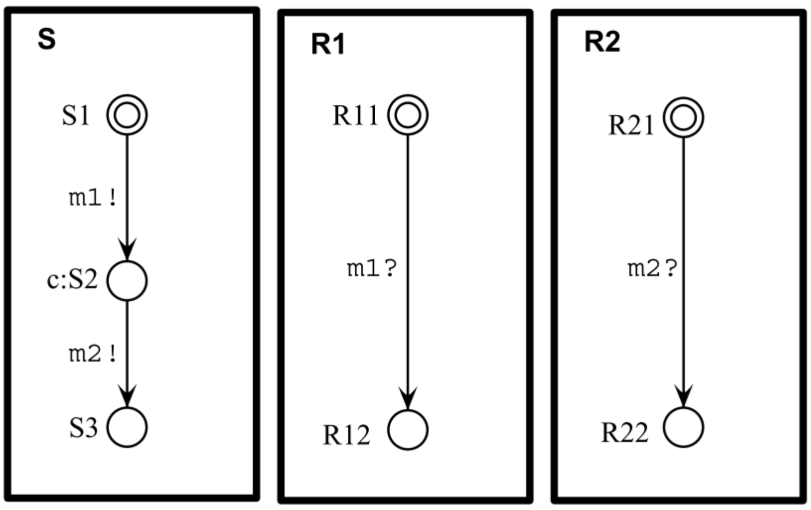


Figure 3-2: Broadcasting communication m_1, m_2 between TA in a network

via input and output actions. In Figure 3-2, S broadcasts either m_1 or m_2 to communicate with $R1$ or $R2$ respectively. Network of Timed Automata (NTA) describe concurrent systems taking into account the synchronous communication. The network is composed using parallel composition. TGA can be extended to a network by parallel composition in the form of Network of Timed Game Automata (NTGA). An NTGA is essentially the synchronised cartesian product of TGA.

Definition 3-1.4 (Network of Timed Game Automata[5])

Let $TGA^i = (L^i, l_0^i, \mathtt{Act}_c^i, \mathtt{Act}_u^i, C^i, E^i, \mathtt{Inv}^i)$ be a timed game automaton for $i \in \{1, ..., n\}$. The parallel composition of $TGA_1, ..., TGA_n$ denoted by $TGA_1|...|TGA_n$ is a timed game automaton $TGA = (L, l_0, \mathtt{Act}_c, \mathtt{Act}_u, C, E, \mathtt{Inv})$ where:

- $L = L^1 \times \cdots \times L^n$
- $l_0 = (l_0^1, \dots, l_0^n)$
- $\mathtt{Act}_c = \{*\} \cup \bigcup_{i=1}^{n} \{a \in \mathtt{Act}_c^i |$ *a is a n internal action*$\}$
- $\mathtt{Act}_u = \{\circledast\} \cup \bigcup_{i=1}^{n} \{a \in \mathtt{Act}_c^i |$ *a is a n internal action*$\}$
- $C = C^1 \cup \cdots \cup C^n$
- *E is defined according to the following rules:*

  - a TGA makes a move on its own via its internal action: the edge is controllable iff the internal action is controllable

  - Two TGAs move simultaneously via a synchronizing action: the edge is controllable iff both input and output actions are controllable (i.e. the environment has priority over the controller)
- $\mathtt{Inv}((l_1, \dots, l_n)) = \mathtt{Inv}^1(l_1) \wedge \cdots \wedge \mathtt{Inv}^n(l_n)$

3-2 Abstraction of Event Triggered Control Systems with Timed Automata

The abstractions as defined in chapter 2 can be represented with a TA in the following manner

Definition 3-2.1 (Traffic Abstraction from [4, 5])

A timed automaton abstracting the triggering times behaviour of a system with triggering coefficient σ is given by $TA^\sigma = (L^\sigma, l_0^\sigma, \mathtt{Act}^\sigma, C^\sigma, E^\sigma, \mathtt{Inv}^\sigma)$ where

- $L^\sigma = \{R_1^\sigma, \dots, R_q^\sigma\}$
- $l_0^\sigma = R_s^\sigma$ *such that* $\xi(0) \in R_s^\sigma$
- $\mathtt{Act}^\sigma = \{*\}$
- $(R_s^\sigma, \underline{\tau}_s^\sigma \leq c \leq \bar{\tau}_s^\sigma, *, \{c\}, R_t^\sigma) \in E^\sigma$ *if* $\mathcal{X}_{[\underline{\tau}_s^\sigma, \bar{\tau}_s^\sigma]}(R_s^\sigma) \cap R_t^\sigma \not\emptyset$
- $\mathtt{Inv}^\sigma(R_s^\sigma) = \{c | 0 \leq c \leq \bar{\tau}_s^\sigma\}$ *for all* $s \in \{1, \dots, q\}$

3-2-1 Scheduling with a Network of Timed Automata

In [5], an approach to scheduling using Networks of Timed Game Automata is proposed. The network is modelled using a Timed Game Automaton with three states: *Idle, InUse and Bad*, and is schematically drawn in Figure 3-3. When one of the connected control loops uses the network, the network jumps from initial state Idle to InUse for a maximum occupation time Δ. If another control loop attempts to use the network while in location *InUse*, the network goes into absorbing state *Bad*. After the Δ time has passed problem-free, the network goes back to *Idle*.

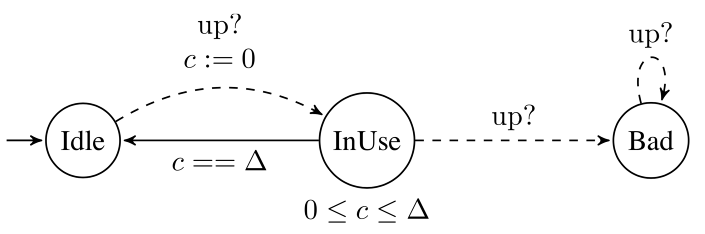


Figure 3-3: A single channel network with synchronising action *up?* [5]

Definition 3-2.2 (Communication Network [5])

Let Δ represent the maximum channel occupancy time, a timed game automation associated with the communication network is given by $TGA^{net} = (L^{net}, l_0^{net}, \mathtt{Act}_c^{net}, \mathtt{Act}_u^{net}, C^{net}, E^{net}, \mathtt{Inv}^{net})$ *where*

- $L^{net} = \{Idle,\ InUse,\ Bad\}$
- $l_0^{net} = Idle$
- $\mathtt{Act}_c^{net} = \{*\}$
- $\mathtt{Act}_u^{net} = \{up?\}$
- $C^{net} = \{c\}$
- E^{net} $=\{$*(Idle, true, up?,*$\{c\}$*, InUse), (InUse,c=*Δ*,*,*$\emptyset$*,Idle), (InUse, true, up?,* $\emptyset$*, Bad), (Bad, true, up?,* $\emptyset$*, Bad)*$\}$
- $\mathtt{Inv}^{net}(InUse) = \{c | 0 \leq c \leq \Delta\}$, $\mathtt{Inv}^{net}(Idle) = \{c | c \geq 0\}$, $\mathtt{Inv}^{net}(Bad) = \{c | c \geq 0\}$

The Timed Gamed Automata for the control loops are formed using the approach from [4], schematically shown in Figure 3-4. For each triggering condition σ_j a separate location $R_s^{\sigma_j}$ is used. To allow for choosing the triggering coefficient and forcing earlier updates, an additional location R_s is included, where the triggering condition is not yet chosen. Location Ear$_s$ is the location that can be used to force an earlier update. When using the event triggering condition, the exact update time cannot be chosen by the controller, it will lie between the upper and lower update time. When activating the earlier update, that exact moment will be the update time. From either a location R^{σ_j} or Ear$_s$, any of the edges can be taken, and will activate uncontrollable action $\{up!\}$. This will trigger the communication networks $\{up?\}$.

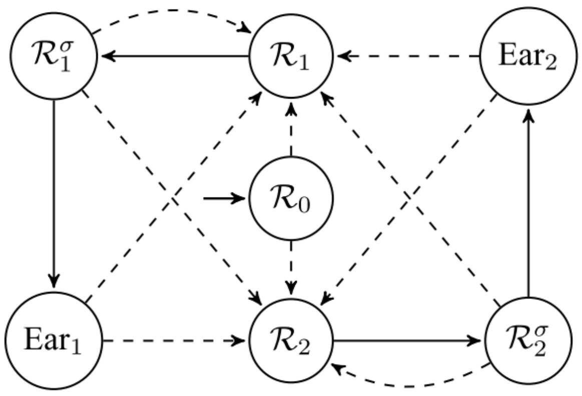


Figure 3-4: A control loop with controllable early updates [5]

Definition 3-2.3 (Control Loop [5])

Consider a set of timed automata $TA^{\sigma_j} = (L^{\sigma_j}, l_0^{\sigma_j}, \mathtt{Act}^{\sigma_j}, C^{\sigma_j}, E^{\sigma_j}, \mathtt{Inv}^{\sigma_j})$ *generated from an event-triggered control loop with triggering coefficient* $\sigma_j \in]0, \bar{\sigma}[$ *for* $j \in \{1, \dots, p\}$ *and assume that* $R_s^{\sigma_1} = \dots = R_s^{\sigma_p}$ *for all* $s \in \{1, \dots, q\}$. *Consider also a set of earlier update time parameters* $\{\underline{d}_1, \bar{d}_1, \dots, \underline{d}_q, \bar{d}_q\}$ *such that*

$$\forall s \in \{1, \dots, q\}\, \exists\, j \in \{1, \dots, p\} : \bar{d}_s \leq \tau_s^{\sigma_j} \tag{3-1}$$

Then, the timed game automata TGA^{cl} *is given by*
$TGA^{cl} = (L^{net}, l_0^{net}, \mathtt{Act}_c^{net}, \mathtt{Act}_u^{net}, C^{net}, E^{net}, \mathtt{Inv}^{net})$ *where*

- $L^{cl} = \bigcup_{j=1}^{p} L^{\sigma_j} \cup \bigcup_{s=1}^{q} \{R_s, Ear_s\}$
- $l_0^{cl} = R_s$ *such that* $\xi(0) \in R_s^{\sigma_j}$
- $\mathtt{Act}_c^{cl} = \mathtt{Act}^{\sigma_1} \cup \bigcup_{j=1}^{p} \{a_j^{cl}\}$
- $\mathtt{Act}_u^{cl} = \{up!\}$
- $C^{cl} = C^{\sigma_1}$
- $E^{cl} = \bigcup_{s=1}^{q} \bigcup_{c \in \mathcal{E}_s} \{(Ear_s, c = 0, up!, \emptyset, R_t)\} \cup \bigcup_{s=1}^{q} \bigcup_{j=1}^{p} \{(R_s, c = 0, a_j^{cl}, \emptyset, R_s^{\sigma_j}),$
$(R_s^{\sigma_j}, \underline{d}_s \leq c \leq \bar{d}_s, *, \{c\}, Ear_s)\} \cup \bigcup_{s=1}^{q} \bigcup_{j=1}^{p} \bigcup_{\{t | (R_s \to R_t) \in E^{\sigma_j}\}}$
$\{(R_s^{\sigma_j}, \underline{\tau}_s^{\sigma_s} \leq c \leq \bar{\tau}^{\sigma_j}, up!, \{c\}, R_t)\}$
- $\mathtt{Inv}^{cl}(R_s^{\sigma_j}) = \{c | c \leq \bar{\tau}^{\sigma_j}\}$, $\mathtt{Inv}^{cl}(R_s) = \{c | c = 0\}$, $\mathtt{Inv}^{cl}(Ear_s) = \{c | c = 0\}$

3-3 Uppaal Stratego

For scheduler design, we use the tool Uppaal Stratego [24] to model our control loop abstractions as defined in Definition 3-2.3 and communications network as defined in Definition 3-2.2.

It can use an efficiënt on-the-fly algorithm for synthesis of reachability and safety objectives as proposed in Uppaal TIGA [25], and can use pricing on hybrid clocks, allowing for the synthesis of objectives with regard to cost under a safety objective.

In the tool Uppaal [6], a *System* models a NTA. Each TA is called a *Process*. *Processes* are instantiated from parametrised *Templates*. Global variables can exist that can be influenced by all *Processes*, and can contain clocks, integer variables, constants and broadcast channels. Broadcast channels can be used to synchronise between different *Processes*. The communication network as described in Definition 3-2.2 is shown in Figure 3-5 A control loop such as

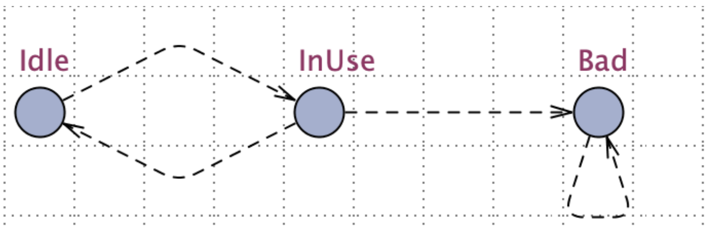


Figure 3-5: A basic communications network drawn in Uppaal Stratego. The guards, invariants, synchronising actions and clock resets have been left out for simplicity

defined in Definition 3-2.3 is drawn in Uppaal Stratego in Figure 3-6

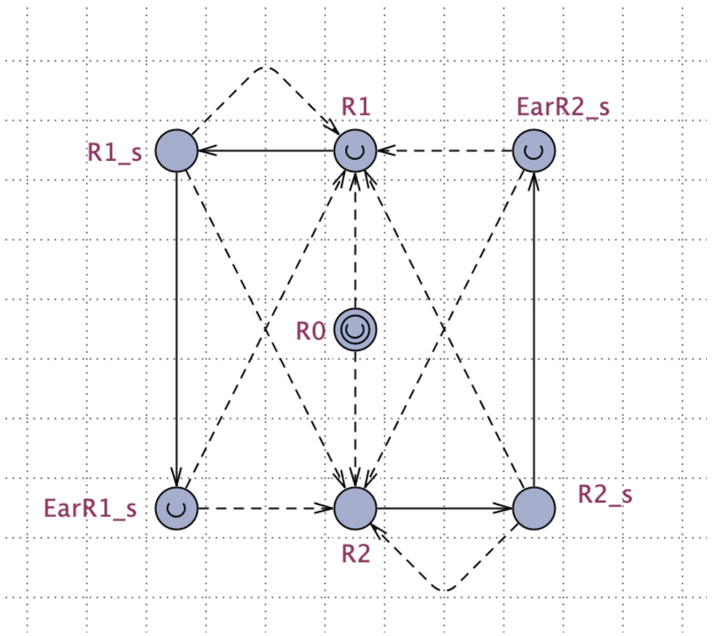


Figure 3-6: A basic control loop drawn in Uppaal Stratego. The guards, invariants, synchronising actions and clock resets have been left out for simplicity

To model an abstraction as a TA, a transition table is needed, and a list of lower- and upper-bound on inter-event time per region. Using this information, a TA defined by Definition 3-2.1 can be generated.

From this abstraction, Control Loops as for example defined by Definition 3-2.3 can be created. Control Loops can be combined with a communications network, for example as defined in

Definition 3-2.2, and the synchronising actions for all components can be set. The entire model now consists of a NTGA.

Chapter 4

Scheduling

In chapter 3 we created a model is ready for use in Uppaal. To find a scheduler using Uppaal Stratego we first define our control objective. Currently, our model is only suited for a synthesis of a safety or reachability objective, since our model does not contain pricing.

4-1 Control Objective and Strategy Generation

The control objective of our scheduler is to avoid communication conflicts while maintaining the individual stability of each connected system. By design, if no communication conflicts take place, the individual systems will remain stable. The *Bad* location of the communications network as defined in Definition 3-2.2n Uppaal, represents a communication conflict. Therefore, the *Bad* location should be avoided. This can be expressed by Uppaal Stratego in the following way:

```
strategy scheduler = control:  A[] not (Network.Bad).          (4-1)
```

Translated as 'create a strategy called scheduler that controls the entire system such that it never reaches the **Bad** location' Avoiding a location, or set of locations is called a *pure safety objective* by Uppaal[26]

4-2 Resulting Strategy

If successful, the synthesis results in a strategy. In this strategy, the initial location is defined, and for each possible location in the Network of Timed Automata (NTA), a number of clock regions is given that require a controllable action, or require to do nothing at that point. This controllable action could for example be forcing an early trigger, or choosing a triggering coefficient for a control loop.

A part of an example strategy for a system with 3 control loops connected to a single communications network looks as following:

```
Initial state:
( cl0.R1 cl1.R1 cl2.R1 Network.Off )
(#time==cl0.c && cl0.c==cl1.c && cl1.c==cl2.c && cl2.c==Network.c0 && Network.c0==0)
Note: The 'strategy' is not guaranteed to be a strategy.

Strategy to avoid losing:




State: ( cl0.R1 cl1.R31 cl2.Ear1 Network.On )
When you are in (67<cl0.c && 10<=cl1.c && cl2.c==5 && cl1.c<=30 && cl2.c==Network.c0
    && Network.c0==5) || (72<cl0.c && cl1.c==5 && 10<=cl2.c && cl1.c==Network.c0
    && cl2.c<=94 && Network.c0==5), take transition cl1.R31->cl1.Ear31 { 30 >= c &&
    5 <= c, tau, 1 }
While you are in    (cl1.c<5 && cl1.c-cl0.c<-67 && cl1.c-cl2.c<=-5 && cl1.c==Network
    .c0 && cl2.c<=94), wait.
```

The output is very verbose, and readable for persons, but is more difficult to directly implement this in software. To convert this text-based output to something more easily used by our software package, we use a *parsing expression grammar*, or *PEG*. *PEG* is a type of analytic grammar, describing a formal language by defining a set of rules. Using a python implementation of a *PEG* parser, we define a grammar for the strategy description. Using this grammar, we can divide that output into locations in the NTA, and parse the clock regions and their corresponding actions.

Chapter 5

Problem Statement

The main goal of this work is to design a tool set that can model a set of control systems communication over a network. The control systems should be modelled with timed automata, where the traffic is abstracted to a timed-automaton. From this modelled set, one should be able to synthesise a scheduler that avoids scheduling conflicts. The synthesised scheduling strategy should be implemented as a scheduler, combined with the continuous system dynamics in a real-time simulation.

Summarizing, we want to design a set of tools that can:

- Abstract event-triggered control systems
- Create a Timed Automaton from this abstraction
- Create a Network of Timed Automata connecting multiple abstractions with a communications network
- Find a scheduling strategy avoiding network conflicts for such a Network of Timed Automata
- Find a cost-optimal scheduling strategy avoiding network conflicts
- Parse the resulting strategy
- Implement the parsed strategy in with a real-time simulation environment to simulate with system dynamics of the control loops

The entire tool chain is schematically shown in Figure 5-1.

After creating the tool set, it will be tested to generate scheduling strategies with an experiment that also has some scientific relevance other than showing the capabilities of the tool set. In order to do this, we will investigate the scalability of the scheduler design approach.

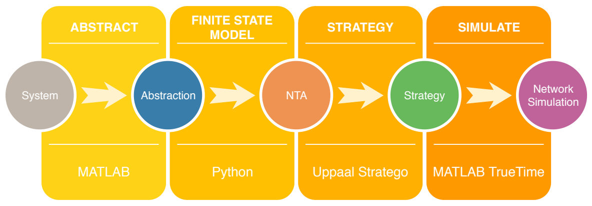


Figure 5-1: Schematic overview of the desired workflow for the entire set of tools from abstraction to simulation

Scalability

To investigate the usability in real-life applications, it is interesting to see how the method to be used scales with increased accuracy of the traffic abstraction, and how it scales with an increasing number of systems used in a network of timed automaton. As mentioned in [19, 27], the number of clocks is a limiting factor in working with timed automata.

The variables measured will be the time necessary to find a strategy, the amount of memory used to find this strategy, and the number of states that need to be visited before the search is complete. The experiments will be discussed more in detail in chapter 7

Chapter 6

Tool Design

In this chapter, the current state of the tool set will be described. The tools are divided into three main categories:

1. Abstractions
2. Timed Automata
3. Strategies

For each category, we describe:

- The languages and packages used
- The concepts that are implemented, and some implementation details

6-1 Abstractions

The implementation of the abstraction approach as described in chapter 2 is based on the code from [4] and [7]. The code allows for modelling 2D and 3D LTI systems. The entire process from creating region matrices Q_s to finding the bounds on inter-event time and a transition table is done in `MATLAB`

6-1-1 Matlab

The tool uses `MATLAB r2018a`. It uses two packages that need to be installed:

- **YALMIP** [28], a toolbox for for modelling and solving optimization problems. This solves the Linear Matrix Inequalities found when calculating the bounds on inter-event time
- **The Multi-Parametric Toolbox (MPT)** [**29**], used for creating *Polyhedra* used in the reachability analysis to create the transition table.

6-1-2 Implementation

After describing the system and the design parameters, the process of abstracting in implemented in a five-step approach by [7]. The five steps are:

1. Finding the global lower bound on the inter-event time. This corresponds to a line-search over τ to find the first value that satisfies Lemma 2-3.1
2. Finding the value of ν, implementing Lemma 2-3.1
3. Finding the lower bound on inter-event time per region $\mathcal{R}_s$ using an adapted golden-section search over σ, implementing Theorem 2-3.1
4. Finding the upper bound on inter-event time per region $\mathcal{R}_s$ using an adapted golden-section search over σ, implementing Theorem 2-3.2
5. Using reachability analysis to create the transition table.

The design parameters are shown in Table 6-1:

Variable	Description	Unit
n	State-space dimension of the abstracted system	[-]
N_conv	Order of the Taylor series approximation of Phi	[-]
sigma_bar	Upper limit for the global lower bound on the inter-event time	[s]
sigma_max	Upper limit for inter-event time	[s]
l_1	Number of subdivisions in the interval [0, sigma_bar]	[-]
l_2	Number of subdivisions in the interval [$\underline{\tau}$,sigma_max]	[-]
m	Number of subdivisions of $\theta \in [0, \pi]$. This should be an even number	[-]
q	Number of regions for half the state space. $q = m^{n-1}$	[-]
alpha	Triggering coefficient	[-]
del_sig	σ' increment	[s]
del_tau_i	Time step size for bound tau in step i of the five steps	[s]
epsilon_tol	Tolerance for the constraint on ε when solving LMI's	[-]
sedumi_eps	Precision for the YALMIP SeDuMi solver	[-]

Table 6-1: Summation of the design parameters used in MATLAB to create abstractions

The first two steps together are straightforward implementations of the functions mentioned. For the third and fourth step, it's interesting to expand on how the line search over σ is implemented using a bi-section search.

Bi-section Search

A bi-section search is a root-finding method for continuous functions known to have two values with opposing signs. It finds the interval in which the zero crossing takes place, switching the function value from one sign to the other by repeatedly bisecting the search interval. At the start, the interval spans the entire search domain. The function f is evaluated at the edges of the interval. We denote the edges of the interval with x_1 and x_3.The interval is split into two sections by probing point $x_2 = \frac{x_3+x_1}{2}$. Either $sign(f(x_1)) = sign(f(x_2))$, indicating the root is in the interval $[x_2, x_3]$, or $sign(f(x_1)) \neq sign(f(x_2))$ and the root is in the interval $[x_1, x_2]$. The new search interval now becomes the interval in which the root is found, and this is repeated until the interval is sufficiently small.

In the case of Theorem 2-3.1 and Theorem 2-3.2, the function being evaluated is an LMI. This function is not unimodal, but either the equations hold ($f(x) = 0$), or they don't ($f(x) = 1$), so at the switching point from $f(x_k) = 0$ to $f(x_{k+1}) = 1$ there is a jump. We are not interested in finding a root, but the value of x where it switches from $f(x) = 0$ to $f(x) = 1$. If chosen properly, the search interval should have only a single switch from 0 to 1 at the bound of inter-event time.

By using an adapted bisection section search, we can find a bound on inter-event time up to a known accuracy within a finite, known number of steps. This number of steps n depends on the size of the initial size of the search interval d and the accuracy ε with the following relation:

$$n = \log_2(\frac{d}{\varepsilon}) \tag{6-1}$$

6-2 Timed Automata

To model the abstractions as Timed Automata, we move away from `MATLAB` to Python. The main reasons to do so is that Python is a freely available language, where `MATLAB` is an highly expensive piece of proprietary software. Also, it is much easier to deal with different versions, where compatibility between different versions of `MATLAB` is more complicated. Scheduling the timed automata will be done in Uppaal Stratego, as this is a tool specifically designed to work with timed automata, and can be later expanded to using pricing information.

6-2-1 Python

Python version 3.6.8 was used to model the abstractions as Timed Automata. The required packages are:

- **shortuuid**. A package to generate unique id's based on the UUID-standard
- **SciPy**. SciPy is used to parse `MATLAB .mat` files, to import the abstractions made with `MATLAB`
- **NumPy**. NumPy is used for several matrix operations performed.
- **PyGraphViz**. PyGraphviz is used to automatically generate a human-readable diagram of Timed Automata using *dot*

6-2-2 Implementation

The following functionality is implemented in the tool:

- Create a Timed Automaton Definition 3-2.1 from a list of lower- and upper bounds combined with a transition table
- Create models for Control Loops such as Definition 3-2.3
- Create models for Communication Networks such as Definition 3-2.2
- Combine Control Loops and Communication Networks into a Network of Timed Automata with appropriate synchronising actions
- Export a Network of Timed Automata to a format readable by Uppaal Stratego

Base

First, a base class for Timed Automata and Timed Game Automata are made, staying close to the formal definitions: sets are defined as python sets and the mapping function is defined as a python dictionary. A base example is shown in Figure 6-1 An edge $l \xrightarrow{g,a,\mathbf{r}} l'$ is defined as a tuple $(l, g, a, \mathbf{r}, l')$ for a timed automaton, as defined in Definition 3-1.2.

```python
"""
NOTE: This is not the actual code used in the tool, but shows a simplified version
    to make it easily understandable
"""

class ta:
    def __init__(self):
        self.locations      = set()
        self.clocks         = set()
        self.actions        = set()
        self.edges          = set()
        self.invariants     = dict()

class tga(ta):
    def __init__(self):
        super().__init__()
        self.actions_u      = set()
        self.actions_c      = set()
        self.actions        = self.actions_u.union(self.actions_c)
```

Figure 6-1: Example for base code for timed automata in Python

Abstraction

Next, we can model the traffic abstraction using this a timed automaton as a base class. The traffic abstraction has a single clock c. We use the transition table to create the locations and the edges between locations: each transition corresponds to an edge.
Next, we assign guards on the transitions based on the list of bounds on inter-event time: each outgoing edge gets a guard with the bounds from the list corresponding to the originating location.
Finally, we create a mapping of invariants to locations using the list of bounds: for each location Rs get an invariant it's clock c up to and including the upper bound on inter-event time: $c <= \bar{\tau}_s$
It should be noted that guards on clocks can only be integer values, and therefore, in the actual tool the time-bounds are scaled by the first power of ten smaller then or equal to the smallest accuracy used in creating the abstraction.

```python
"""
NOTE: This is not the actual code used in the tool, but shows a simplified version
    to make it easily understandable
"""

class abstraction(ta):
    def __init__(self, bounds, transitions):
        super().__init__()
        self.clocks.update({'c'})
        # Create locations from number of bounds and create invariants
        for n, bound in enumerate(bounds):
            self.locations.update({f'{n}'})
            lower_bound, upper_bound = bound
            self.invariants.update({f'{n}': f'c<={upper_bound}'})

        # Create edges from transition tables and bounds
        # Note: currently, transitions is not a matrix but a list of lists
        # For each location, there is a list with indices of reachable locations
        # We add the guard and reset the clock 'c'
        for index, i in enumerate(transitions):
            for j in transitions[i]:
                lower_bound, upper_bound = bounds[index]
                guard = f'{lower_bound} < c && {c <= upper_bound}'
                edge = (f'{index}',guard, None, 'c', f'{j}')
                self.edges.update(edge)
```

Figure 6-2: Example of a base abstraction in Python

Control Loop Model

Now the basics are set. We create a class that models a control loop corresponding to Definition 3-2.3 based on an abstraction as created in python as shown above. Our control loop distinguishes between controllable and uncontrollable actions, and will use the timed game automaton class as a base.
In this example, we will not model a choice in triggering coefficient. It is possible however, to force an early update.
We consider triggering in the natural triggering interval to be an uncontrollable action, so an edge for a natural trigger will be an uncontrollable edge. Forcing an early update is controllable, but the region we will transition to is unknown, and the edge representing it is therefore uncontrollable.
As defined in Definition 3-2.3, we allow early updates in a window of time of d seconds before the start of the natural triggering interval.

The design variables that are used can be defined for a control loop are show in Table 6-2 and creating a control loop in Python is shown in Figure 6-3

Variable	Description	Unit
sync	the broadcast-channel used to synchronise timed automata in a network	[-]
d	size of the early trigger interval, scaled accordingly.	[-]
initial_location	list of possible initial locations	list

Table 6-2: Summation of the design parameters used in the control loop model

Communications Network Model

To model a communications network, the model as described in Definition 3-2.2 is chosen. The network has only uncontrollable edges, and is synchronised over broadcast channel "up?". The network will stay in the $InUse$ location for $delta$ seconds.

```python
"""
NOTE: This is not the actual code used in the tool, bu shows a simplified version to for demonstration
"""

class ControlLoop(tga):
    def __init__(abstraction, sync='up', d=5):
        super().__init__()
        self.sync = sync
        # We copy the clocks from the abstraction ta
        self.clocks.update(abstraction.clocks)

        # We create a set of locations we want to mark as urgent: no time may pass in these locations
        self.urgent = set()

        # We copy the locations from the abstraction ta and create early triggering locations
        # We mark each early trigger as urgent and add a controllable edge to each early trigger
        for location in abstraction.locations:
            self.locations.update({f'R{location}'})
            self.urgent.update({f'Ear{location}'})
            guard = f""
            edge = (f'R{location}',guard, internal_action, False, clocks, f'Ear{location}')
            self.edges.update(edge)
        self.locations.update(self.urgent)


        # We copy the invariants from the abstraction ta
        self.invariants.update(abstraction.invariants)

        # We convert the edges according to the transition table from (early) locations
        for edge in abstraction.edges:
            self.edges.update({self.uncontrollable(edge)})
            self.edges.update({self.early(edge)}


        def uncontrollable(self, edge):
            """     Convert an edge from (l,g,a,c,l') -> (l,g,a_c,a_u,c,l')     """
            (start, guard, internal_action, clocks, end) = edge
            return f'R{start}', guard, internal_action, frozenset({f'{self.sync}!'}), clocks, f'R{end}'

        def controllable(self, edge):
            """     Convert an edge from (l,g,a,c,l') -> (l,g,a_c,a_u,c,l')     """
            (start, guard, internal_action, clocks, end) = edge
            return f'R{start}', guard, internal_action, False, clocks, f'R{end}'

        def early(self, edge)
            """     Convert an edge from (l,g,a,c,l') -> (l,g,a_c,a_u,c,l')     """
            (start, guard, internal_action, clocks, end) = edge
            return f'R{start}', guard, internal_action, frozenset({f'{self.sync}!'}), clocks, f'R{end}'
```

Figure 6-3: A control loop as a Timed Automaton in Python

Network of Timed Automata

Breaking the trend with earlier models, for the network of timed automata, for a Network of Timed (Game) Automata, we no longer follow the formal definition. Keeping in mind the way Uppaal models a network of timed automata, we simply keep a list of the Timed (Game) Automata that are included in the network. An example is shown in Figure 6-5
When used in Uppaal, simply instantiating the list of Timed (Game) Automata with a synchronising action will create the entire network.

Exporting to Uppaal

When the Network of Timed Automata is modelled, we want to export it to Uppaal Stratego to do further analysis. Uppaal Stratego uses XML-files to describe Timed Automata and the

```python
"""
NOTE: This is not the actual code used in the tool, but shows a simplified version
    for demonstration purposes
"""
class Network(TGA):
    def __init__(sync="up", delta="5")
        # We add the three locations
        self.locations.update({"Idle", "InUse", "Bad"})
        self.clocks = {"c"}

        # We add an uncontrollable edge from Idle to InUse on "up?" resetting "c"
        edge = ("Idle", True, False, f'{sync}?', 'c=0', "InUse")
        self.edges.update({edge})

        # We add an uncontrollable edge from InUse to Bad on "up?"
        edge =("InUse", True, False, f'{sync}?', 'c=0', "Bad")
        self.edges.update({edge})

        # We add an uncontrollable edge from Bad to Bad on "up?"
        edge =("Bad", True, False, f'{sync}?', 'c=0', "Bad")
        self.edges.update({edge})

        # We add an uncontrollable edge from InUse to Idle with guard "c==delta"
        edge =("InUse", f'c=={delta}', False, None, 'c=0', "Idle")
        self.edges.update({edge})

        # We create an invariant for InUse, where we can stay up to delta seconds
        invariant = {"InUse": f'c <= {delta}'}
        self.invariants.update(invariant)
```

Figure 6-4: Code example of a class describing a Communications Network

```python
"""
NOTE: This is not the actual code used in the tool, but shows a simplified version
    to make it easily understandable
"""

class nta:
    def __init__(*ta):
        self.ta = ta
```

Figure 6-5: A Network of Timed Automata as modelled in Python

necessary declarations. To export our model to a XML-file that can be read by Uppaal, we use a modified version of pyuppaal [30].
Pyuppaal was created to make Uppaal models from commandline and has support for a graphical editor as well. It was created for Python 2.7 and some dependencies can not easily be upgraded to Python 3.
The part in pyuppaal that exports (Networks of) Timed Automata to XML has been updated to be used with Python 3, and the rest has been stripped. As Uppaal sometimes has difficulties with multiple locations being on the same physical location in the 2D state-space that defines visualisation of a timed automaton, support for auto-layout is also kept.
Using mixed inheritance, timed automata are turned into *Templates* for Uppaal, and a network of timed automata is turned into a *System*. Some extra flexibilty is added to the Uppaal classes. Binding it all together, the network of timed automata can now be exported to an XML-file for Uppaal.

Uppaal from Python

Uppaal has a graphical user interface, but we are not interested in that for uses other than visual validation of the model. We can also approach Uppaal from the command-line interface (CLI), using the `verifyta` command. Python allows for system calls to the CLI via the *subprocess* module, and in our demo it is shown how this works. For more information on `verifyta`, run `verifyta -h` from the CLI of your system running Uppaal.

6-3 Strategies

Uppaal Stratego can output two types of strategies:

- TIGA Strategies
- Stratego Strategies

The strategy as seen in section 4-2 is a TIGA strategy, a strategy with a pure safety objective. A pure safety objective is a problem of reachability, and doesn't include any type of pricing or stochastics.
A Stratego strategy can be an optimization with regards to price or probability. As we have not yet implemented pricing or probability, we focus on TIGA Strategies. A TIGA strategy uses a lot of text, but has a clear structure. We use this structure to retrieve the valuable information using a technique based on *parsing expression grammar (PEG).*

6-3-1 Python

Python version 3.6.8 was used to model the abstractions as Timed Automata. The required packages are:

- Parsimonious [31] Parsimonious is an arbitrary-lookahead parser based on PEGs

6-3-2 Implementation

Parsimonious can parse text based on a defined grammar, dividing text into nodes. Next, each node can be visited, and a action to be taken can be defined. Since the output for TIGA is automatically generated, it has a relatively simple structure, and this can be described using such a grammar. We consider a part of a strategy shown in Figure 6-6 as an example. The strategy state can be split into two parts:

```
State: ( cl0.R1 cl1.Ear28 cl2.Ear37 Network.Off )
While you are in    (67<cl0.c && 5<=cl1.c && cl1.c-cl2.c<=-5 && cl2.c<=24) || (67<
     cl0.c && 5<=cl2.c && cl1.c<=40 && cl2.c<=24 && cl2.c-cl1.c<=-5), wait.
When you are in (67<cl0.c && 35<cl1.c && cl2.c==5 && cl2.c==Network.c0 && Network.c0
     ==5) || (67<cl0.c && 10<=cl1.c && cl2.c==5 && cl1.c<35 && cl2.c==Network.c0 &&
     Network.c0==5) || (72<cl0.c && cl1.c==5 && 10<=cl2.c && cl1.c==Network.c0 &&
     cl2.c<=94 && Network.c0==5), take transition Network.On->Network.Off { c0 == 5,
      tau, 1 }
```

Figure 6-6: Example of a single state in TIGA output

- Locations
- Clock zones

For a combination of locations, one for each timed automaton in the network of timed automata, a number of clock regions is given.
For each clock region, an action is shown. There are two kinds of actions: 'take a transition' or 'wait'. Clock regions with the same action are bundled together. We use this knowledge to create a bit of grammar to parse the text. This could look like Figure 6-7.

```
TIGAGrammer = Grammar(
r"""
state           = st_open locations action
st_open         = newline+ "State:" ws*
locations       = "(" location+ ws ")"
action          = (move / delay)+
"""
)

class TIGAParser(NodeVisitor):
    grammar = TIGAGrammar

    def visit_state(self, node, visited_children):
        st_open, locations, actions = visited_children
        print(locations)

    def visit_action(self, node, visited_children):
        actions = visited_children
        for action in actions:
            print(action.expr_name)

    def generic_visit(self, node, visited_children):
        return visited_children
```

Figure 6-7: Example of a bit of grammar to parse a state, with a node visitor class to process the nodes that are created

The entire grammar for parsing TIGA output is found in the project repository on GitHub [32]. After defining a grammar to split the text into nodes, a node visitor can be made that visits each node and create an appropriate action. Using this, we can for example parse the strategy and save it in the form of a binary decision diagram or a timed automaton.

Chapter 7

Scalability

Small scale experiments have proven to work well, but for wide scale application of timed automata in scheduling, it's important to see how it scales with respect to accuracy of the abstractions, and to the number of systems connected. Therefore, a we will test our tool by setting up two experiments to test the scalability and discuss their results.

7-1 Scaling of Timed Automata

At each point in time the possible future behaviour of a Network of Timed Automata (NTA) is determined by its active locations and the values of all its clocks [19]. In Uppaal, this is considered to be the state. Although there are uncountable many of these states, in [19] it is shown that through an equivalence relation, these states can be mapped to a finite number of clock zones. The number of clock zones scales linearly with the number of locations, but exponentially with the number of clocks. The reachability problem we are trying to solve is known to be `PSPACE`-complete in [19].

7-2 Experiments

We are interested in the scaling with respect to accuracy of the abstraction, visible in the number of regions $\mathcal{R}_s$ the state-space has been divided in, and with respect to the number of systems present in the NTA. As the number of clocks increases linearly with the number of systems, this is expected to have unwanted results. For each experiment, we will measure the number of states Uppaal visits before finding a *winning* strategy, the amount of *CPU time* it takes and the amount of internal memory is used. This is measured using Uppaals internal measuring program based on *memtime*, a small utility to measure time and memory consumption on POSIX OSes

We will do both experiments with (combinations of) the same systems modelled in the case study (chapter 4) of [5]. The first control loop is given by:

$$\begin{aligned}\dot{\xi} &= \begin{bmatrix} 0 & 1 \\ -2 & 3 \end{bmatrix} \xi + \begin{bmatrix} 0 \\ 1 \end{bmatrix} v, \\ v &= \begin{bmatrix} 1 & -4 \end{bmatrix} \xi. \end{aligned} \tag{7-1}$$

The second control loop is given by:

$$\begin{aligned}\dot{\xi} &= \begin{bmatrix} -0.5 & 0 \\ 0 & 3.5 \end{bmatrix} \xi + \begin{bmatrix} 1 \\ 1 \end{bmatrix} v, \\ v &= \begin{bmatrix} 1.02 & -5.62 \end{bmatrix} \xi. \end{aligned} \tag{7-2}$$

Each experiment is characterized by two parameters:

- The number of conic regions of a system$q$
- The number of systems connected to the communications network

The number of consecutive updates is limited: `EarMax = 4`. The systems both have a triggering coefficient of $\sigma = 0.05$, and the early update can be updated from 0.005 time units before the lower bound on the triggering time. The maximum consecutive number of updates `EarMax = 4`.

For both experiments, we generate a model from the abstractions based on the definition of a control loop in Definition 3-2.3. Control Loops are denoted `cli` with i the identifying number for a control loop. The communications network is denoted `Network`. The control objective is: `control: A[] not (Network.Bad)`

7-2-1 Increasing accuracy

To measure the performance when increasing the accuracy of the abstractions, we increase the number of subdivisions m from 20 to 180 in steps of 20. This results in a number of regions $q = 2 \times m^{n-1} = 2 \times m$ ranging from 40 to 360 in steps of 40.

For each run we connect two systems to the communications network, one instantiation of the first control loop, and one instantiation of the second control loop.

7-2-2 Increasing number of systems

To measure the performance when increasing the number of clocks, we increase the number of systems connected to the communications network gradually. We alternate in the addition of a control loop to the network: First a control loop based on an abstraction of Equation 7-1 is added, then a control loop based on an abstraction of Equation 7-2 is added. If the number of control loops is increased again, a loop similar to the first one is added, next one similar to the second, and so on. The number of regions per control loop is kept constant at $q = 8$

7-3 Results

7-3-1 Increasing Accuracy

First, we take a look at the way the abstraction scales as the accuracy becomes larger. Theoretically, with our current modelling approach, the number of edges per control loop $N_{e,cl}$ can maximally become

$$\begin{aligned} N_{e,cl} &= 2q^2 + q \\ \frac{N_{e,cl}}{q} &= 2q + 1 \end{aligned} \tag{7-3}$$

growing quadratically. In worst case, the average number of edges per region grows linearly. As shown in Figure 7-1, the number of edges and edges per regions indeed grow respectively quadratically and linearly, but stays far away from the worst case growth in 7-3.

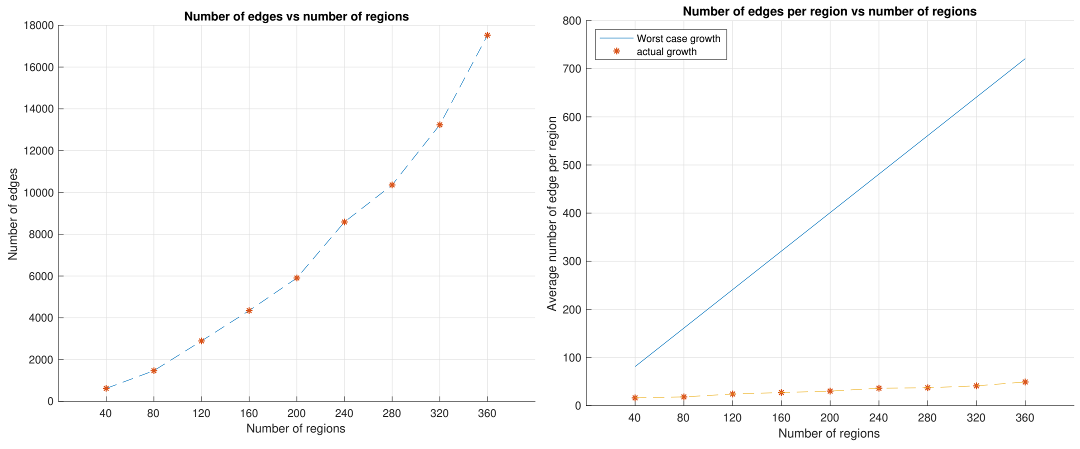


Figure 7-1: Plotting the number of edges in a control loop versus the number of regions in the abstraction (left) and the average number of edges per region versus the number of regions (right). In the right, the worst case growth is also plotted.

In Figure 7-3 the increase of calculation versus number of regions and number of edges respectively is shown. As expected, the number of states grows linearly with the number edges. The number of states grows quadratically with the number of regions, which is also to be expected as the number of edges grows quadratically with the number of regions.

Figure 7-4 shows the increase of memory consumed. The results are similar to the number of states stored, which makes sense as most of the memory is consumed by storing the states. In Figure 7-2, a box plot is shown for the computation time and memory consumption, to show the variance in the experiments. For the number of states stored, no plot is made. This number is a deterministic quantity, and doesn't change with different runs of the experiment.

In Figure 7-5 the increase of calculation time with respect to the number of regions and number of edges is shown. When increasing the accuracy, the time used seems to increase irregularly. This can be because the calculation time is also dependent on the number and size of clock zones. As the data doesn't easily fit a pattern, the only conclusion to be drawn

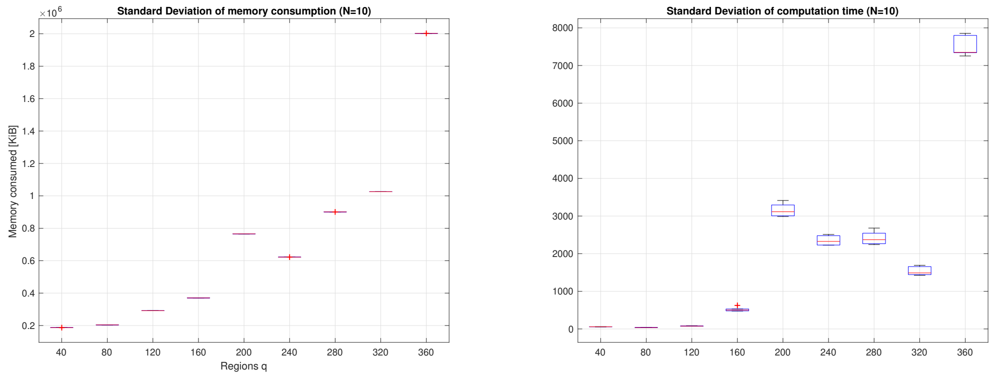


Figure 7-2: A box plot showing the difference between runs.

is that as the number of regions, the time necessary to find a solution is increasing with rapid paces, but will depend on the particular abstraction.

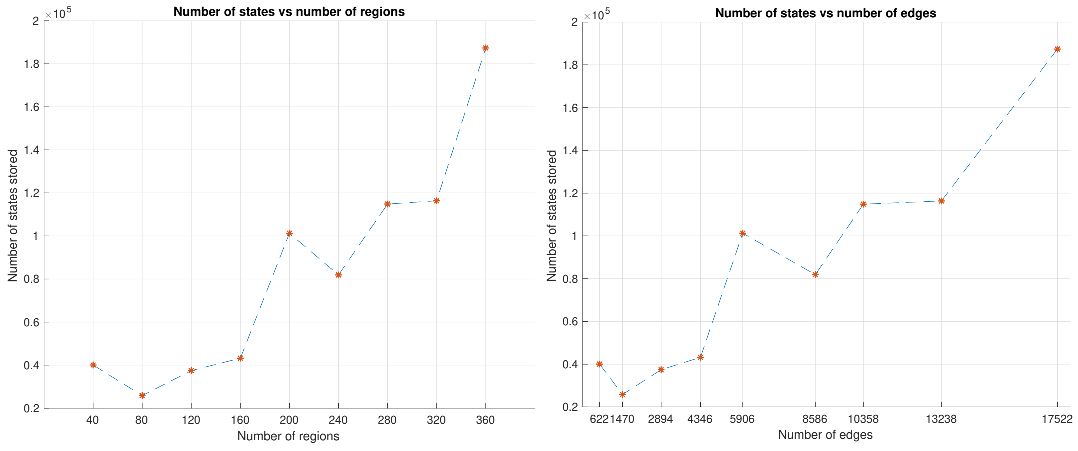


Figure 7-3: The number of states stored in memory to finding a 'winning' strategy versus an increasing number of regions and edges.

7-3-2 Increasing number of systems

When increasing the number of systems, things scale very badly. Even with an abstraction with a very small number of regions, and therefore edges, it was only possible to find a solution up to three control loops connected to the communication network. When trying to add a fourth, the search was cancelled after 5 days, already having consumed almost 10 GiB of memory. In Figure 7-6, the results for up to three control loops is shown, averaged over 10 runs. Even with only three data points, any extrapolation of the data would suggest an enormous increase in all three variables.

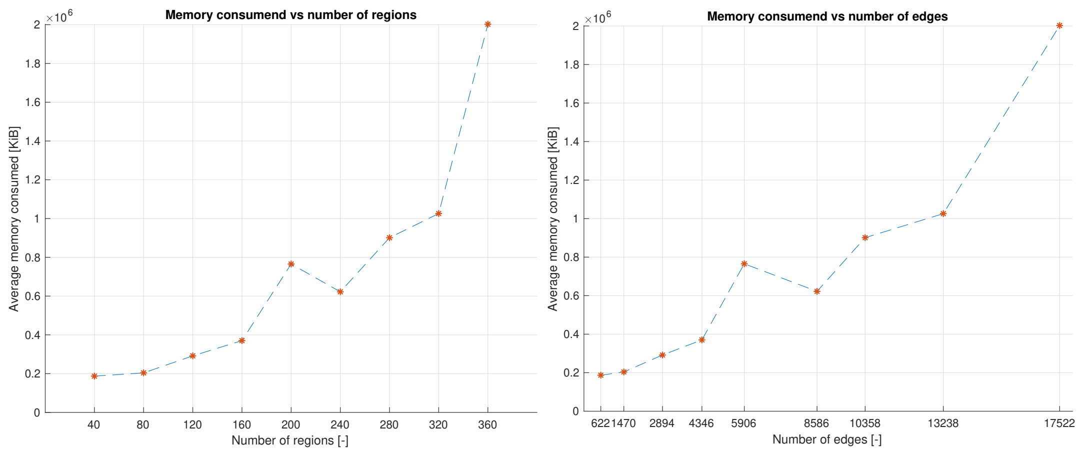


Figure 7-4: The average memory consumption in for finding a 'winning strategy versus an increasing number of regions and edges. Average over 10 runs

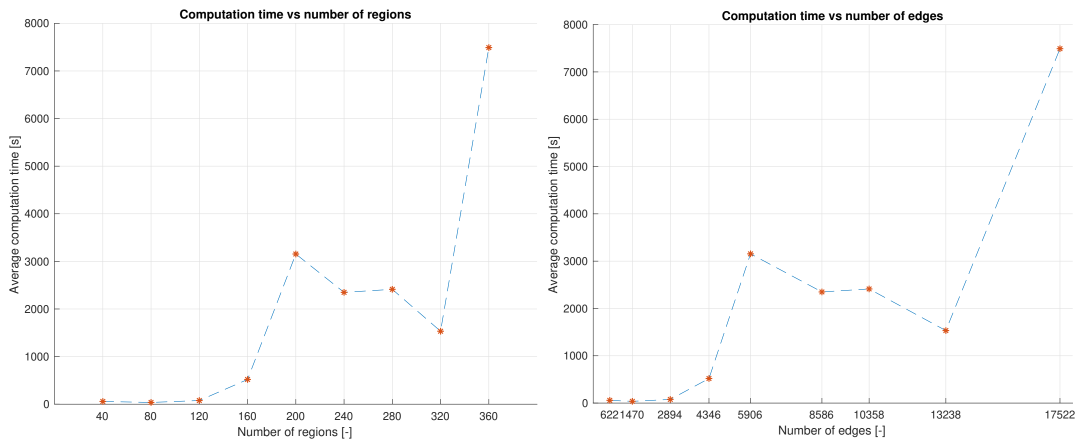


Figure 7-5: The average computation time to finding a 'winning' strategy versus an increasing number of regions and edges. Average over 10 runs

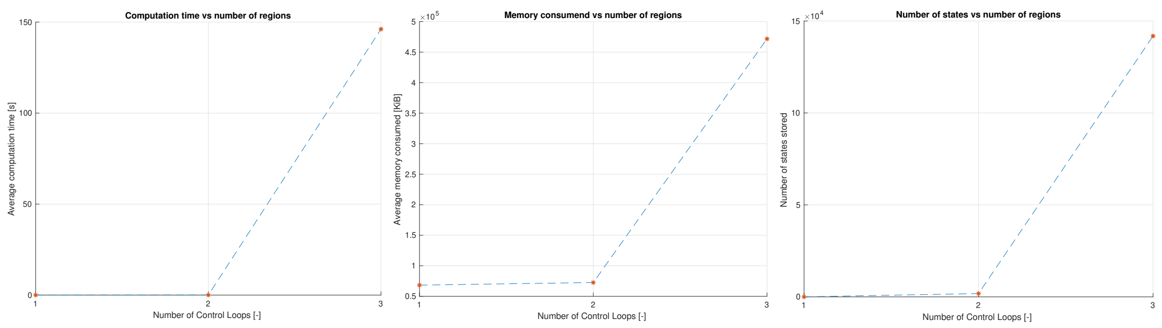


Figure 7-6: The computation time, memory consumed and number of states stored to finding a 'winning' strategy versus an increasing number of control loops q

Chapter 8

Conclusion and Recommendations

In this chapter, we first conclude on the design criteria set in the problem statement, and give recommendations for continuing the development in the future. Next, we review the outcome of the scalability experiments, and discuss some paths that can be taken next to either improve scalability, or to work with the modelling as is.

8-1 Tool design

8-1-1 Conclusion

In chapter 5, the design criteria of a set of tools was set. Several, but not all design criteria have been met, as shown in Table 8-1.

Abstract event-triggered control systems	✓
Create a Timed Automaton from this abstraction	✓
Create a Network of Timed Automata connecting multiple abstractions with a communications network	✓
Find a scheduling strategy avoiding network conflicts for such a Network of Timed Automata	✓
Find a cost-optimal scheduling strategy avoiding network conflicts	×
Parse the resulting strategy	–
Implement the parsed strategy in with a real-time simulation environment to simulate with system dynamics of the control loops	×

Table 8-1: Table displaying design criteria that have been met

To nuance the parsing of resulting strategies, grammar for parsing strategies has been created. However, to implement the strategy, further extension of the parsing might be necessary.
The abstraction tools in `MATLAB` as used by [7] have been improved to be more efficiënt, but is still restricted to using 2D and 3D systems.

For working with timed automata, a new tool in Python has been made. There has not been any work done on simulating the system dynamics controlled by a network scheduler driven by the strategy generated in Uppaal.

8-1-2 Future work

Future work would be to first fill in the missing items from the design criteria, pricing and simulation, to complete the set of tools.
Next, the abstraction process could be ported to Python, removing `MATLAB` from the list of required software. When moving the code to Python, it would be wise to re-think the implementation of some parts of code. Although the code is in line with the math backing it up, there might be smarter ways of implementing some parts. This could result in a significant speed-up when creating abstractions. Specifically, finding an initial estimate of a global upper bound on the inter-event time would simplify finding the upper bound. Furthermore, new abstraction principles could be integrated into the tools, and their performance could be compared to the current models.

8-2 Scalability

8-2-1 Conclusion

In the scalability experiments, it is shown that more accurate abstractions can also lead to a state-space explosion. When more systems are added to the Network of Timed Automata, increasing the number of clocks with each added system, the generation of a scheduling strategy quickly becomes impractical. Using this approach for more than three systems connected to a communications network is not feasible as is.

8-2-2 Recommendations

First, we recommend to try removing the clock from the communications network, reducing the number of clocks in the system. This can be done by creating an extra synchronising command *down*. Just as in the modelling in Definition 3-2.3, a transition triggers the network using the *up* broadcast channel. Now, the triggering system will remain in a temporary state for a time Δ before going to the next location, broadcasting over the *down* channel. An example of this is given in Figure 8-1.
Secondly, to increase the usability of the proposed approach, combining multiple schedulers in a round-robin manner could be modelled. For example, continuing on the model in Figure 8-1, one could reintroduce a clock to the Communication Network. Using this clock, we can now define a time interval Δ_1 in which we allow triggering, and a time interval Δ_2 in which the network is unavailable. Say we want to connect 9 control loops to a single network. We divide the Communications Network in a repeating sequence of intervals of Δ_1 in size, making $\Delta_2 = 2\Delta_1$. We model the Network in three instances as shown in Figure 8-2, with three control loops connected to each communications network. This splits the problem into 3 problems of 3 control loops, instead of 1 problem with 9 control loops. This way, we

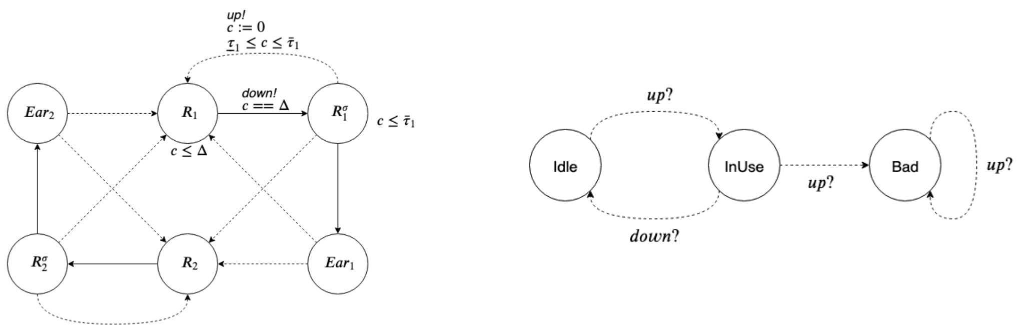


Figure 8-1: An schematic drawing of a Control Loop (left) with an extra broadcast channel $down$, and a communications network without an internal clock (right)

might avoid the state-space explosion caused by too many clocks in a single network of timed automata.

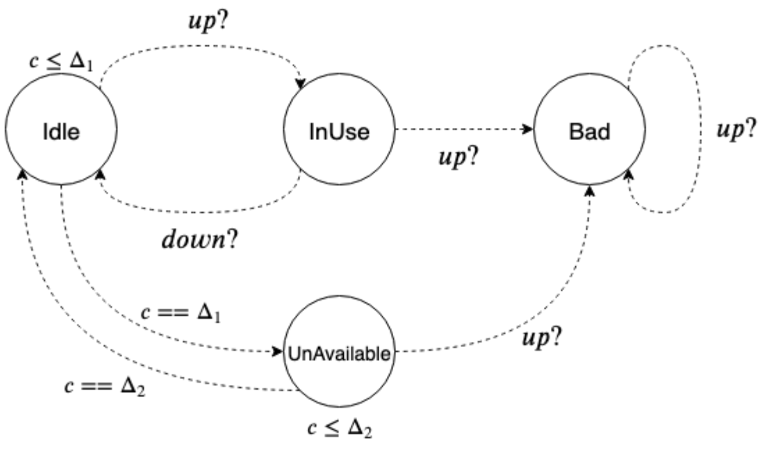


Figure 8-2: A communications network that is usable for a time period of Δ_1, and unavailable for a period of Δ_2

Bibliography

[1] T. Yang, "Networked control system: a brief survey," *IEE Proc. - Control Theory Appl.*, vol. 153, pp. 403–412, jul 2006.

[2] W. Heemels, "Event-Triggered and Self-Triggered Control."

[3] S. Al-Areqi, D. Gorges, S. Reimann, and S. Liu, "Event-based control and scheduling codesign of networked embedded control systems," in *2013 Am. Control Conf.*, vol. 45, pp. 5299–5304, IEEE, jun 2013.

[4] A. S. Kolarijani and M. Mazo, "Formal Traffic Characterization of LTI Event-Triggered Control Systems," *IEEE Trans. Control Netw. Syst.*, vol. 5, no. 1, pp. 274–283, 2018.

[5] D. Adzkiya and M. Mazo, "Scheduling of Event-Triggered Networked Control Systems using Timed Game Automata," oct 2016.

[6] G. Behrmann, A. David, and K. G. Larsen, "A Tutorial on Uppaal 4.0," tech. rep., 2006.

[7] C. Hop, *Abstraction of In-Vehicle Event- Triggered Networked Control Systems for Scheduling.* Thesis, Delft University of Technology, 2017.

[8] A. Chutinan and B. Krogh, "Computing polyhedral approximations to flow pipes for dynamic systems," vol. 1, no. December, pp. 2089–2094, 2002.

[9] W. Heemels, K. Johansson, and P. Tabuada, "An Introduction to Event-triggered and Self-triggered Control," *2012 IEEE 51st IEEE Conf. Decis. Control*, pp. 3270–3285, dec 2012.

[10] K. Astrom and B. Bernhardsson, "Comparison of Riemann and Lebesgue sampling for first order stochastic systems," in *Proc. 41st IEEE Conf. Decis. Control. 2002.*, vol. 2, pp. 2011–2016, IEEE.

[11] P. Tabuada and S. Member, "Event-Triggered Real-Time Scheduling of Stabilizing Control Tasks," vol. 52, no. 9, pp. 1680–1685, 2007.

[12] M. Velasco, J. M. Fuertes, and P. Martí, “The Self Triggered Task Model for Real-Time Control Systems,” in *Proc. 24th Real-Time Syst. Symp.*, pp. 67–70, 2003.

[13] H. Rehbinder and M. Sanfridson, “Scheduling of a limited communication channel for optimal control,” in *Proc. 39th IEEE Conf. Decis. Control (Cat. No.00CH37187)*, vol. 1, pp. 1011–1016, IEEE, 1996.

[14] S. Longo, G. Herrmann, and P. Barber, “Optimization Approaches for Controller and Schedule Codesign in Networked Control,” *IFAC Proc. Vol.*, vol. 42, no. 6, pp. 301–306, 2009.

[15] A. Cervin and P. Alriksson, “Optimal On-Line Scheduling of Multiple Control Tasks: A Case Study,” in *18th Euromicro Conf. Real-Time Syst.*, pp. 141–150, IEEE, 2006.

[16] M.-M. Ben Gaid, A. Cela, and Y. Hamam, “Optimal Real-Time Scheduling of Control Tasks With State Feedback Resource Allocation,” *IEEE Trans. Control Syst. Technol.*, vol. 17, pp. 309–326, mar 2009.

[17] S. Al-Areqi, D. Gorges, and S. Liu, “Robust control and scheduling codesign for networked embedded control systems,” in *IEEE Conf. Decis. Control Eur. Control Conf.*, vol. 45, pp. 3154–3159, IEEE, dec 2011.

[18] S. Al-Areqi, D. Gorges, and S. Liu, “Stochastic event-based control and scheduling of large-scale networked control systems,” *2014 Eur. Control Conf. ECC 2014*, pp. 2316–2321, 2014.

[19] R. Alur and D. L. Dill, “A theory of timed automata,” *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

[20] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, “HYTECH: A Model Checker for Hybrid Systems,” tech. rep.

[21] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, “TIMES-A Tool for Modelling and Implementation of Embedded Systems,” tech. rep.

[22] C. Fiter, L. Hetel, W. Perruquetti, and J.-P. Richard, “A state dependent sampling for linear state feedback,” *Automatica*, vol. 48, pp. 1860–1867, 2012.

[23] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, “Symbolic Model Checking for Real-Time Systems,” *Inf. Comput.*, vol. 111, pp. 193–244, jun 1994.

[24] A. David, P. G. Jensen, K. G. Larsen, M. Mikučionis, and J. H. Taankvist, “LNCS 9035 - Uppaal Stratego,” *Baier C., Tinelli C. Tools Algorithms Constr. Anal. Syst.*, vol. 9035, pp. 206–211, 2015.

[25] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime, “Uppaal-Tiga: Timed Games for Everyone,” tech. rep.

[26] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime, “Uppaal Tiga User-manual,” tech. rep.

[27] G. Behrmann, J. Bengtsson, A. David, K. G. Larsen, P. Pettersson, and W. Yi, “UppaaL Implementation Secrets,” pp. 3–22, 2002.

[28] J. Lofberg, “YALMIP : a toolbox for modeling and optimization in MATLAB,” pp. 284–289, 2005.

[29] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari, “Multi-Parametric Toolbox 3.0,” tech. rep.

[30] M. C. Olesen, “PyUppaal,” 2008.

[31] E. Rose, “Parsimonious,” 2019.

[32] P. Schalkwijk, “Project Github Repository,” 2019.

Acronyms

CORA	Cost-Optimal Reachability Analysis
DUT	Delft University of Technology
ETC	Event-triggered Control
LTI	Linear Time-Invariant
LMI	Linear Matrix Inequalities
LPTA	Linear Priced Timed Automata
NCS	Networked Control Systems
NTA	Network of Timed Automata
NTGA	Network of Timed Game Automata
PTA	Priced Timed Automata
PTGA	Priced Timed Game Automata
SMC	Statistical Model Checking
SPTGA	Stochastic Priced Timed Game Automata
STC	Self-triggered Control
TA	Timed Automata
TGA	Timed Game Automata
TCTL	Timed Computation Tree Logic
TSA	Timed Safety Automatons

Appendix A

Calculating the region matrices and vertices

From [7], Chapter 3.3, 3.4.

A-1 Region Matrix Calculations

Consider a region R_1 characterized by the angles $\theta_{min} = 0\text{rad}$ and $\theta_{max} = \frac{\pi}{4}$. For each angle, consider a line through the origin at an angle $\theta_{min}, \theta_{max}$ respectively. Both these lines divide the state space into two half-spaces. They can be defined by their normal vector as $a_i^T x = 0$ where $x = [x_1 x_2]^T$. From both these lines the normal vector a_i pointing to the inside of the region R_1 is drawn. The half-spaces to which these normal vectors point are defined by $a_i^T x \geq 0$. The normal vectors are calculated as:

$$a_1^T = [\,\sin(\theta_{min}) cos(\theta_{min})] \tag{A-1}$$

$$a_2^T = [\sin(\theta_{max}) cos(\theta_{max})] \tag{A-2}$$

The intersection of the two half-spaces is exactly the region R_1. Since for states that lie within the half-spaces it holds that:

$$a_1^T x \geq 0$$

and:

$$a_2^T x \geq 0$$

for the states that lie within the intersection of the half-spaces (which is R_1) it holds that:

$$x^T (a_1 a_2^T) x \geq 0$$

Instead of defining the matrices Q_s (corresponding to the regions R_s) as $Q_s = (a_1 a_2^T)$ they are defined as:

$$Q_s = (a_1 a_2^T + a_2 a_1^T)$$

In this way the matrices Q_s are made symmetric, which is numerically advantageous since the Linear Matrix Inequality (LMI) solvers that are used to calculate the sample time bounds for the regions R_s can handle symmetric matrices more efficiently compared to general matrices.

A-2 Calculating Region Vertices

The region polyhedra in which the state space is partitioned can be defined by their vertices. Calculating these vertices is useful for the reachability analysis. For a two-dimensional system, where each region is defined by one angular coordinate, the vertices can be calculated as:

$$\begin{aligned} x_1 &= \cos(\theta) \\ x_2 &= \sin(\theta) \end{aligned} \tag{A-3}$$

for both the minimum and maximum angle (θ_{min} and θ_{max}) for that region, resulting in two different vertices. For an n-dimensional systems (with $n \geq 3$), each region is defined by $(n-1)$ angular coordinates. With a minimum and maximum value for each angular coordinate, an n-dimensional region is defined by $2^{(n-1)}$ vertices. The coordinates for each vertex V can be calculated as:

$$\begin{cases} x_1 = \cos(\theta_1) & \text{if } |\theta_2| \neq \frac{\pi}{2} \\ x_2 = \sin(\theta_1) & \text{if } |\theta_2| \neq \frac{\pi}{2} \\ x_1 = x_2 = 0 & \text{if } |\theta_2| = \frac{\pi}{2} \\ x_3 = |x_2| \tan(\theta_2) & \text{if } |\theta_2| \neq \frac{\pi}{2} \\ x_3 = 1 & \text{if } \theta_2 = \frac{\pi}{2} \\ x_3 = -1 & \text{if } \theta_2 = \frac{\pi}{2} \\ \vdots & \\ x_{(i+1)} = |x_i| \tan(\theta_i) & \text{if } |\theta_i| \neq \frac{\pi}{2} \\ x_{(i+1)} = 1,\ x_i = x_{(i-1)} = \ldots = x_1 = 0 & \text{if } \theta_i = \frac{\pi}{2} \\ x_{(i+1)} = -1,\ x_i = x_{(i-1)} = \ldots = x_1 = 0 & \text{if } \theta_i = -\frac{\pi}{2} \\ \vdots & \\ x_n = |x_{n-1}| \tan(\theta_{n-1}) & \text{if } |\theta_{(n-1)}| \neq \frac{\pi}{2} \\ x_n = 1,\ x_{(n-1)} = x_{(n-2)} = \ldots = x_1 = 0 & \text{if } \theta_{(n-1)} = \frac{\pi}{2} \\ x_n = -1,\ x_{(n-1)} = x_{(n-2)} = \ldots = x_1 = 0 & \text{if } \theta_{(n-1)} = -\frac{\pi}{2} \end{cases} \tag{A-4}$$

where $\theta_1 \in [0, \pi]$ and $\theta_i \in [\frac{-\pi}{2}, \frac{\pi}{2}]$ for $i \in \{2, ..., (n-1)\}$. Note that by considering all angular coordinates over an interval of length π only half of the state space is considered. With each of the $n-1$ angular coordinates (lying within an interval of length π) divided into m subintervals, this half of the state space will contain $m^{(n1)}$ conic regions. Due to the aforementioned symmetry the regions in the first half of the state space can be mapped to the second half of the state space by taking $V = V$ for each vertex V . The entire state space then is divided into $2 \times m^{(n-1)}$ conic regions. Calculating the vertices in this way is consistent with the region representation as given in Equation 2-12