ETCetera

Beyond Event-Triggered Control

Delimpaltadakis, Giannis; De Albuquerque Gleizer, Gabriel; Van Straalen, Ivo; Mazo, Manuel

DOI
10.1145/3501710.3519523

Publication date
2022

Document Version
Final published version

Published in
Proceedings of the 25th ACM International Conference on Hybrid Systems (HSCC 2022)

ETCetera: beyond Event-Triggered Control

Giannis Delimpaltadakis*
i.delimpaltadakis@tudelft.nl
Delft University of Technology
Delft, The Netherlands

Gabriel de A. Gleizer*
g.gleizer@tudelft.nl
Delft University of Technology
Delft, The Netherlands

Ivo van Straalen*
ivo65@live.nl
Delft University of Technology
Delft, The Netherlands

Manuel Mazo Jr.
m.mazo@tudelft.nl
Delft University of Technology
Delft, The Netherlands

ABSTRACT

We present ETCetera, a Python library developed for the analysis and synthesis of the sampling behaviour of event triggered control (ETC) systems. In particular, the tool constructs abstractions of the sampling behaviour of given ETC systems, in the form of timed automata (TA) or finite-state transition systems (FSTSs). When the abstraction is an FSTS, ETCetera provides diverse manipulation tools for analysis of ETC's sampling performance, synthesis of communication traffic schedulers (when networks shared by multiple ETC loops are considered), and optimization of sampling strategies. Additionally, the TA models may be exported to UPPAAL for analysis and synthesis of schedulers. Several examples of the tool's application for analysis and synthesis problems with different types of dynamics and event-triggered implementations are provided.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Theory of computation** → *Abstraction*; • **Networks** → Cyber-physical networks;

KEYWORDS

event-triggered control, networked control systems, abstraction, scheduling

ACM Reference Format:
Giannis Delimpaltadakis, Gabriel de A. Gleizer, Ivo van Straalen, and Manuel Mazo Jr.. 2022. ETCetera: beyond Event-Triggered Control. In *25th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '22), May 4–6, 2022, Milan, Italy.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3501710.3519523

1 INTRODUCTION

Since the seminal works of Åström [5] and Årzen [1], and in particular since the work of Tabuada [40], there has been a surge of interest in the use of event-based controller update techniques in order to reduce the resource consumption of (networked) control systems. Despite the myriad of proposals made in the literature, very little attention has been given, until recently, to the quantification of the benefits of such proposals. As such, little was known about the sampling patterns arising from so called event-triggered control (ETC) systems, what is their efficiency (e.g. what is the average inter-sample time), or how hard are they to schedule.

In the past decade this gap in the literature was addressed in a number of papers. Both analytical [36, 37] and computational approaches [12–16, 18, 20, 21, 32, 35] have been pursued to study ETC's sampling behaviour. Particularly, the computational approach builds on the notion of abstractions. These abstractions distil the sampling-times' dynamics from ETC's hybrid dynamical model, and serve as a computationally tractable object which may be employed to address a wide range of verification and synthesis questions on ETC's sampling patterns. For their construction, the state-space of the control system is partitioned in an informed way. The resulting abstraction is either a timed-automaton (TA), or in the case of periodic ETC (PETC) a finite transition system (FTS), capturing the sampling times' dynamics. We call these objects *traffic models*, evoking how they model the communication traffic (of measurements and actuation commands) generated by the control system.

Once such traffic models are available they can be employed for two main purposes: *analysis* of metrics such as average inter-sample times measuring the efficiency of an ETC design [14, 16]; or for *synthesis* of scheduling strategies [35], or optimized sampling strategies [12]. Synthesizing scheduling strategies is a necessary capability when implementing event triggered control systems sharing a communication medium. The aim of the schedulers synthesized is to prevent multiple control-loops (or other real-time tasks) requesting a channel access simultaneously, which would lead to communication collisions and potentially to control instability. One may achieve such scheduling goal by enabling earlier transmissions than dictated by the ETC mechanism, or even by allowing switching between different triggering conditions. In a similar way, the traffic models can be employed to optimize sampling strategies by enforcing, at times, earlier sampling of the control loop than that dictated by ETC. The rationale in this optimization is that while one is generally interested in minimizing the *cumulative* or *average* amount of communications employed, ETC approaches are greedy in trying to optimize such goals, i.e. they try to enlarge as much as possible each individual inter-sample time interval. In general, greedy optimization approaches are suboptimal, and as such early

*Giannis Delimpaltadakis, Gabriel de A. Gleizer and Ivo van Straalen contributed equally to this work.

HSCC '22, May 4–6, 2022, Milan, Italy

ACM ISBN 978-1-4503-9196-2/22/05.
https://doi.org/10.1145/3501710.3519523

triggering can be used as a control variable to optimize beyond what the greedy ETC strategies can attain.

All of these developments: model construction, analysis, and synthesis of schedulers or sampling strategies, are automated techniques that rely on heavy computation. In the present paper we describe a toolbox that implements all these developments. The tool is named ETCetera and is available at https://gitlab.tudelft.nl/sync-lab/ETCetera. It allows the user to:

(1) Construct traffic models of nonlinear ETC systems with arbitrary triggering conditions and disturbances;
(2) Construct specialized traffic models of Linear PETC systems with Quadratic triggering conditions;
(3) Export models to UPPAAL [11] for scheduler synthesis;
(4) Synthesize schedulers effectively for several PETC systems sharing a channel;
(5) Synthesize optimized sampling strategies;
(6) Analyze PETC systems through the computation of traffic metrics;

In what follows, we provide a brief introduction to event-triggered control and other needed preliminaries (Section 2), describe the steps involved in the abstractions the tool constructs (Sections 3.1 and 3.2), present the approach implemented in ETCetera for PETC scheduler design (Section 3.3), and briefly introduce other synthesis and analysis problems also implemented in the tool (section 3.4). Finally, Section 4 provides the basic instructions of use of the tool, which are then illustrated on several examples.

2 PRELIMINARIES

2.1 Event-triggered control

Consider a control system with state-feedback:

$$\dot{\zeta}(t) = f(\zeta(t), u(\zeta(t)), d(t))$$

where t is time, $\zeta(t) \in \mathbb{R}^{n_\zeta}$ is the state, $u(t) \in \mathbb{R}^{n_u}$ is the control input, $d(t) \in \mathbb{R}^{n_d}$ is an unknown signal (e.g., a disturbance) and $f : \mathbb{R}^{n_\zeta} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_d} \to \mathbb{R}^{n_\zeta}$ is the vector field. In *sample-and-hold* control implementations, the input is held constant between consecutive *sampling times* t_i, t_{i+1}:

$$\dot{\zeta}(t) = f(\zeta(t), u(\zeta(t_i)), d(t)), \quad t \in [t_i, t_{i+1}) \tag{1}$$

Specifically, in *continuous event-triggered control* (CETC), sampling times are defined as follows:

$$t_{i+1} = t_i + \min\left\{\tau_{\max}, \inf\{t > 0 : \phi(\zeta(t), \zeta(t_i)) \geq 0\}\right\} \tag{2}$$

where $t_0 = 0$, (2) is called the *triggering condition*, $\phi(\cdot, \cdot)$ is called the *triggering function*, and the difference $t_{i+1} - t_i$ is called the *inter-sample time*. The constant $\tau_{\max}$ is a forced upper bound on inter-sample times, and it prevents the system from running open-loop for an indefinite amount of time. Between two consecutive sampling times t_i and t_{i+1}, the triggering function starts from a negative value $\phi(\zeta(t_i), \zeta(t_i)) < 0$, and remains negative until it becomes zero at t_{i+1}^-; at t_{i+1}^+, the state $\zeta(t_{i+1})$ is sampled again, the control input is updated $u(\zeta(t_{i+1}))$, the triggering function resets to a negative value, and the process is repeated again. As the inter-sample time depends solely on the initial condition $\zeta(t_i)$ and the unknown signal $d(t)$, we denote it by:

$$\tau_r(x) := \min\Big\{\tau_{\max}, \inf\{t > 0 : \phi(\zeta(t), \zeta(t_i)), \zeta(t_i) = x, d(t) = r(t)\}\Big\} \tag{3}$$

and when $d(t) = 0$ (e.g., unperturbed systems) we just write $\tau(x)$.

CETC owes its name to the fact that the triggering condition is checked continuously by the sensors, to detect *events* (moments when the triggering function becomes zero) right when they happen. In contrast to this regime, in a special variant of ETC termed *periodic ETC* (PETC), the triggering condition is only checked periodically in time. In such cases, the sampling times are defined as:

$$t_{i+1} = t_i + \min\left\{k_{\max} h, \inf\{kh : k \in \mathbb{N}, \phi(\zeta(kh), \zeta(t_i)) \geq 0\}\right\} \tag{4}$$

where h is the *checking period* and $k_{\max} \in \mathbb{N}$ again determines an upper bound on inter-sample times.

2.2 Transition systems

The ETC traffic models built by the ETCetera tool can be encapsulated as *generalized transition systems*, following the definition from Tabuada in [41]:

Definition 1 (Transition system). *A system $\mathcal{S}$ is a tuple $(\mathcal{X}, \mathcal{X}_0, \mathcal{U}, \mathcal{E}, \mathcal{Y}, H)$ where:*

- *$\mathcal{X}$ is the set of states,*
- *$\mathcal{X}_0 \subseteq \mathcal{X}$ is the set of initial states,*
- *$\mathcal{U}$ is the set of actions,*
- *$\mathcal{E} \subseteq \mathcal{X} \times \mathcal{U} \times \mathcal{X}$ is the set of edges (or transitions),*
- *$\mathcal{Y}$ is the set of outputs, and*
- *$H : \mathcal{X} \to \mathcal{Y}$ is the output map.*

A (transition) system is said to be *finite* if $\mathcal{X}$ and $\mathcal{U}$ are finite; this is the case of *automata*, for which efficient tools exist for verification of properties and synthesis of controllers. A close relative, but infinite-state, for which such tools also exist is a *timed automaton* [3], in which a finite system is equipped with a finite set of *clocks* c_i whose values can only vary according to $\dot{c}_i = 1$ or be reset on edges. A transition on a system is denoted by $x \xrightarrow{u} x'$ meaning that $(x, u, x') \in \mathcal{E}$. An infinite *run* of a system is a sequence $x_0 \xrightarrow{u_0} x_1 \xrightarrow{u_1} x_2 \ldots$, and the set of all possible runs of $\mathcal{S}$ is called its *internal behavior set*. Every run maps into an *external behavior*, or *trace* $y_0 u_0 y_1 u_1 y_2 \ldots$ by taking $y_i = H(x_i)$. The set of all possible traces (or *infinite external behavior set*) of $\mathcal{S}$ is denoted by $\mathcal{B}_\mathcal{S}$.

An *abstraction* is a simpler version of a system preserving some of its properties. For verification, an abstraction needs to preserve the external behavior set $\mathcal{B}_\mathcal{S}$ (possibly including additional "spurious" behaviors), while for control synthesis it needs to, loosely writing, preserve the behaviors that can be *enforced* by the controller through its action set. The associated formal notions are, respectively, *(bi)simulations* and *alternating (bi)simulations*. See [41] for the formal definitions and associated methods. In this work, ETC systems and their sampling behaviors are abstracted by finite transition systems or timed automata; in this way, the constructed abstractions are used either to verify properties (verification) of a given ETC system's behavior or to synthesize traffic schedulers

(control synthesis) that avoid packet collisions in networks shared by multiple ETC loops.

3 THEORETICAL UNDERPINNINGS

3.1 Traffic abstractions of CETC systems with disturbances

The ETCetera tool constructs abstractions that capture the sampling behavior of a given ETC system: *traffic models*. In the general case, where the given system (1)-(2) may be a nonlinear CETC system with disturbances or uncertainties and the triggering function is of general form, the methods used are the ones developed in [20, 21]. For the construction of these abstractions, among some technical assumptions (see [21]), we assume the following:

Assumption 1.

- *The system operates in a bounded hyper-rectangular state-space* $X \subset \mathbb{R}^{n_\zeta}$ *(parameter* `state_space_limits`*).*
- *The unknown signal* $d(t)$ *is bounded by a hyper-rectangular domain* $D \subset \mathbb{R}^{n_d}$ *(parameter* `disturbance_limits`*):* $d(t) \in D$ *for all* t*.*

The constructed abstraction is a transition system $\mathcal{S} = (\mathcal{X}, \mathcal{X}_0, \mathcal{E}, \mathcal{Y}, H)$ where:

- $\mathcal{X}_0 = \mathcal{X} := \{\mathcal{R}_i : i \in \mathcal{I}\}$, where $\mathcal{I}$ is a finite set of indices (e.g., $\mathcal{I} = \{1, 2, \ldots, q\}$), and $\bigcup_{i\in\mathcal{I}} \mathcal{R}_i = X$; i.e., the sets $\mathcal{R}_i$ constitute a partition of the ETC system's state space.
- $\mathcal{Y} \subseteq 2^{\mathbb{R}^+}$ and $H(\mathcal{R}_i) = [\underline{\tau}_{\mathcal{R}_i}, \overline{\tau}_{\mathcal{R}_i}]$, where:

$$\begin{aligned} \underline{\tau}_{\mathcal{R}_i} &\leq \inf_{x\in\mathcal{R}_i,\ d(t)\in D} \tau_d(x) \\ \overline{\tau}_{\mathcal{R}_i} &\geq \sup_{x\in\mathcal{R}_i,\ d(t)\in D} \tau_d(x) \end{aligned} \tag{5}$$

- $(\mathcal{R}_i, \mathcal{R}_j) \in \mathcal{E}$, if $\exists\, x \in \mathcal{R}_i$ and $t \in H(\mathcal{R}_i)$ s.t. $\zeta(t; x) \in \mathcal{R}_j$.

We have omitted the set of actions $\mathcal{U}$, since the constructed abstractions are autonomous and they are only meant to abstract the sampling behavior of the given system (for how these abstractions may later be endowed with actions, to control the system's sampling and schedule ETC network traffic, see Section 3.3). Each state of the abstraction $\mathcal{R}_i$ represents a region in the state space of the ETC system. Observe that the output of state $\mathcal{R}_i$ is an interval containing all possible inter-sample times that may be exhibited by the system if initialized in $\mathcal{R}_i$ (under any admissible realization of the unknown signal $d(t)$). Due to these observations and how the transitions have been defined, it can be seen that: for any sequence of inter-sample times $\tau_1, \tau_2, \ldots$ that the ETC system (1)-(2) may exhibit, there is a corresponding trace of the abstraction $[\underline{\tau}_{\mathcal{R}_1}, \overline{\tau}_{\mathcal{R}_1}], [\underline{\tau}_{\mathcal{R}_2}, \overline{\tau}_{\mathcal{R}_2}], \ldots$ such that $\tau_i \in [\underline{\tau}_{\mathcal{R}_i}, \overline{\tau}_{\mathcal{R}_i}]$ for all i. Thus, the abstraction captures the ETC system's sampling behavior. In fact, one can establish formally that the abstraction *ϵ-approximately simulates* the ETC system's sampling behavior (see [32, 35]). What is more, as elaborated in [35], it is semantically equivalent to a *timed automaton* (e.g., see [3]), where the intervals $[\underline{\tau}_{\mathcal{R}_i}, \overline{\tau}_{\mathcal{R}_i}]$ appear appropriately in the automaton's *guards* and *invariants*. As such, once the traffic model has been constructed, the numerous algorithms that have been developed for timed automata (e.g., [11]) may be used for verification and/or traffic scheduler synthesis.

As described in [21], the intervals $[\underline{\tau}_{\mathcal{R}_i}, \overline{\tau}_{\mathcal{R}_i}]$ are computed via an iterative algorithm combining a line-search and reachability analysis computational tools (dReach [33] and Flow* [9]). Specifically, we run a line-search over the variables $\underline{\tau}_{\mathcal{R}_i}$ and $\overline{\tau}_{\mathcal{R}_i}$ until the following are verified by the reachability analysis tools:

$$\begin{aligned} &\bullet\ \phi(\zeta(t; x), x) < 0, \quad \forall t \in [0, \underline{\tau}_{\mathcal{R}_i}], \forall x \in \mathcal{R}_i, \forall d(t) \in D \\ &\bullet\ \phi(\zeta(\overline{\tau}_{\mathcal{R}_i}; x), x) \geq 0, \forall x \in \mathcal{R}_i, \forall d(t) \in D \end{aligned} \tag{6}$$

Indeed, as shown in [20, 21], the above conditions can be transformed to set-membership properties of the ETC system's trajectories, and thus checked by dReach or Flow*. As soon as the line-search finds parameters $\underline{\tau}_{\mathcal{R}_i}$ and $\overline{\tau}_{\mathcal{R}_i}$ for which (6) holds, the algorithm stops, as (6) implies (5). Observe that, in this way, the determined intervals are generally larger than the tightest possible intervals dictated by (5). Larger intervals imply more non-determinism in the abstraction (the abstraction approximates the actual sampling behavior more coarsely).

The transitions $\mathcal{E}$ are also computed via reachability analysis, since the condition for transitions presented above may also be transformed to set-membership properties of the system's trajectories. Note that the computations performed by reachability analysis tools are not exact (e.g. the sets and the trajectories are overapproximated); this may lead to *spurious transitions* (transitions not exhibited by the actual system's behavior), further contributing to the abstraction's non-determinism.

Moreover, the tool supports two different ways of partitioning the state space X into regions $\mathcal{R}_i$:

Grid: Creating a grid over X via hyper-rectangles $\mathcal{R}_i$. The user has to specify parameters $(p_1, p_2, \ldots, p_{n_\zeta})$ (`grid_points_per_dim`), such that X is partitioned into $p_1 \times p_2 \times \cdots \times p_{n_\zeta}$ equal hyper-rectangles $\mathcal{R}_i$.

IM-based: Partitioning X via approximations of *isochronous manifolds*[1] (IMs) and cones, and obtaining $\mathcal{R}_i$ as the sets that are delimited by these approximations and cones. In particular, due to theoretical subtleties, the state space partitioning is done slightly differently, depending on the dynamics:

- *Homogeneous Systems with* $d(t) = 0$: When the given system is *homogeneous* (for the definition, see [22]) and $d(t) = 0$, the process is as follows. To compute approximations of isochronous manifolds M_{t_k}, the user has to specify the times $(t_1, \ldots, t_k, \ldots, t_q)$ (parameter `manifolds_times`, in ascending order) to which the manifolds correspond. To determine the cones C_j, we follow a similar process to the *isotropic covering* of [26], in which each i-th angular spherical coordinate of the state-space is partitioned into a_i equal angles. Thus, the user has to specify the parameters $(a_1, \ldots, a_j, \ldots, a_{n_\zeta - 1})$ (`angles_discretization`), to obtain at most[2] $a_1 \times a_2 \times \ldots a_{n_\zeta - 1}$ cones C_j. Finally, the regions $\mathcal{R}_i$ are obtained as intersections between cones C_j, sets between consecutive approximations of isochronous manifolds M_{t_k} and $M_{t_{k+1}}$

[1] Isochronous manifolds M_{t_k} are sets of points in the system's state-space that correspond to the same inter-sample time t_k; i.e., the system initialized anywhere in M_{t_k} exhibits the same inter-sample time t_k (under the same realization of the unknown signal $d(t)$). For more information, see [22, 23].

[2] Some of the constructed cones are discarded, because they are degenerate (Lebesgue measure-zero). Nonetheless, the union of all remaining non-degenerate cones constitutes a partition of $\mathbb{R}^{n_\zeta}$.

and the set X. Moreover, since the regions $\mathcal{R}_i$ admit a representation that is complex for the employed reachability analysis tools, we overapproximate them via ball-segments contained inside the cones C_j. Then, dReach is used for reachability analysis, as it can handle ball-segments efficiently. An illustration of IM-based partitioning for homogeneous systems is shown in Fig. 1.

- *Non-homogeneous Systems with $d(t) = 0$:* When the system is *non-homogeneous* and $d(t) = 0$, it is lifted to $\mathbb{R}^{n_\zeta+1}$ by adding a dummy variable w, where it is rendered homogeneous. The state-space X of the original system is now mapped to the set $\tilde{X} = \{(x, 1) \in \mathbb{R}^{n_\zeta+1} : x \in X\}$. The user has to specify parameters $(p_1, p_2, \dots, p_{n_\zeta})$ (`grid_points_per_dim`), such that $\tilde{X}$ is partitioned into $p_1 \times p_2 \times \cdots \times p_{n_\zeta}$ equal hyper-rectangles. Each of these hyper-rectangles determines a polyhedral cone C_j pointed at the origin, with its extreme vertices being the vertices of the hyper-rectangle. Furthermore, as in the previous case, the times $(t_1, \dots, t_k, \dots, t_q)$ (parameter `manifolds_times`, in ascending order) have to be given, to derive approximations of isochronous manifolds M_{t_k}. Finally, the regions $\mathcal{R}_i$ are obtained as intersections between cones C_j, the sets between consecutive approximations of isochronous manifolds M_{t_k} and $M_{t_{k+1}}$ and the set $\tilde{X}$. As above, the regions $\mathcal{R}_i$ are overapproximated by ball-segments contained in C_j, and reachability analysis is carried out via dReach. An illustration of IM-based partitioning for non-homogeneous systems is shown in Fig. 2.
- $d(t) \neq 0$: In this case, even though the theory developed in [21] does address it, IM-based partitioning is not yet supported by ETCetera, due to the fact that dReach cannot handle disturbances (and Flow* cannot handle ball-segments). We are working towards addressing this issue in future versions of the tool.

Finally, the approximations of IMs are computed with the iterative algorithm described in [22], which employs the SMT solver dReal [27]. The user may specify dReal's precision parameter δ (`precision_deltas`, with default value 10^{-7}), which affects the accuracy of approximation; the smaller δ is, the more accurate the approximations are, but the computational load becomes larger.

Generally, grid-partitioning is more computationally efficient w.r.t. the computational load of creating the abstraction. On the other hand, IM-based partitioning generally provides tighter intervals $[\underline{\tau}_{\mathcal{R}_i}, \overline{\tau}_{\mathcal{R}_i}]$, thus reducing one of the abstraction's sources of non-determinism. However, the current implementation of the tool does not check if the given `manifolds_times` result into approximations of isochronous manifolds which cover the whole state-space of interest X; thus, in IM-based partitioning, the user has to inspect the end-result to verify that the state-space is indeed covered by the regions $\mathcal{R}_i$. Finally, we note that even when grid-partitioning is employed, the tool uses approximations of isochronous manifolds to derive estimates on timing lower bounds $\underline{\tau}_{\mathcal{R}_i}$ and use them as a starting-point for the line-search algorithms, thus speeding up computations. For more information on the abstractions' construction and on approximations of isochronous manifolds the reader is referred to [20, 21] and [22, 23], respectively.

3.2 Traffic abstractions of linear PETC systems

In the case of unperturbed linear systems under PETC, the construction of abstractions can be performed with simpler computations,

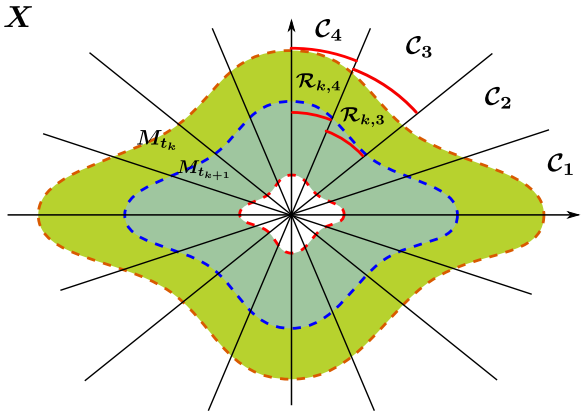


Figure 1: IM-based partitioning for a homogeneous CETC system. The rays passing through the origin determine the cones C_j. The dashed lines are consecutive approximations of IMs M_{t_k} and $M_{t_{k+1}}$. The regions $\mathcal{R}_{k,j}$ are delimited by consecutive approximations of IMs and cones (hence the indexing). The circular segments in cones C_3 and C_4 delimit the ball segments that overapproximate regions $\mathcal{R}_{k,3}$ and $\mathcal{R}_{k,4}$.

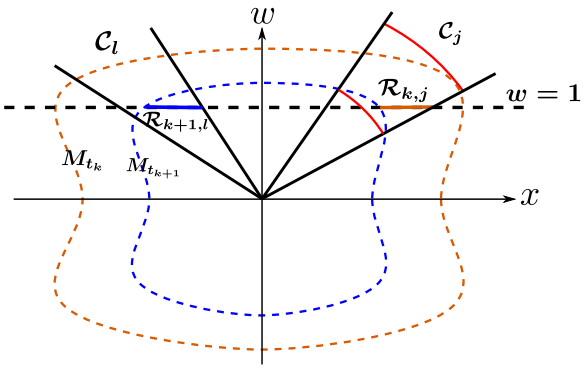


Figure 2: IM-based partitioning for a non-homogeneous CETC system, which has been embedded in $\mathbb{R}^{n_\zeta+1}$. The mapped state-space $\tilde{X}$ is a subset of the $w = 1$-plane. The rays passing through the origin determine the polyhedral cones C_j. The dashed curves are consecutive approximations of IMs M_{t_k} and $M_{t_{k+1}}$. The regions $\mathcal{R}_{k,j}$ are intersections between cones C_j, the sets between consecutive approximations of isochronous manifolds M_{t_k} and $M_{t_{k+1}}$ and the set $\tilde{X}$ (hence the indexing). The circular segments in cone C_j delimit the ball segment, say $\mathcal{B}_{k,j}$, which defines an overapproximation of $\mathcal{R}_{k,j}$ as follows: $\mathcal{R}_{k,j} \subseteq \mathcal{B}_{k,j} \cap \tilde{X}$.

provide exact simulations (thanks to the periodic nature of condition checking), and in general the abstractions are tighter than when employing the general scheme described in the previous section.

Consider the PETC system (1),(4) where the plant is a linear system without disturbances

$$\dot{\zeta}(t) = A\zeta(t) + BK\zeta(t_i)), \quad t \in [t_i, t_{i+1}),$$

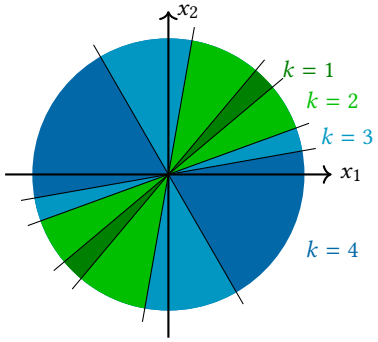


Figure 3: An example of isochronous subsets $\mathcal{R}_k$ for a two-dimensional linear PETC system.

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are the plant matrices and $\boldsymbol{K}$ is the controller gain. Suppose further that the triggering function is quadratic, i.e., $\phi(\boldsymbol{\zeta}(t), \boldsymbol{\zeta}(t_i)) = \begin{bmatrix} \boldsymbol{\zeta}(t) \\ \boldsymbol{\zeta}(t_i) \end{bmatrix}^{\mathsf{T}} \boldsymbol{Q} \begin{bmatrix} \boldsymbol{\zeta}(t) \\ \boldsymbol{\zeta}(t_i) \end{bmatrix}$. Such quadratic triggering functions include the most common ones in the literature [29]. This case has some special properties that allow for tighter abstractions than in the nonlinear case. First, from any sampled state one can determine exactly the state reached after k:

$$\begin{aligned} \boldsymbol{\zeta}(t) &= \boldsymbol{M}(kh)\boldsymbol{\zeta}_{\boldsymbol{x}}(t_i), \ \forall t \in [t_i, t_{i+1}] \\ \boldsymbol{M}(t) &:= \mathrm{e}^{\boldsymbol{A}t} + \int_0^t \mathrm{e}^{\boldsymbol{A}\tau}\mathrm{d}\tau \boldsymbol{B}\boldsymbol{K} =: \boldsymbol{A}_{\mathrm{d}}(t) + \boldsymbol{B}_{\mathrm{d}}(t)\boldsymbol{K}. \end{aligned} \tag{7}$$

An *isochronous subset*, which is the part of the state space that triggers at a given inter-sample time k, can be implicitly represented by a conjunction of quadratic inequalities [13]:

$$\mathcal{R}_k := Q_k \setminus \left(\bigcap_{j=1}^{k-1} Q_j \right) = Q_k \cap \bigcap_{j=1}^{k-1} \bar{Q}_j,$$

$$Q_k := \begin{cases} \{\boldsymbol{x} \in \mathbb{R}^{n_\zeta} \mid \boldsymbol{x}^{\mathsf{T}}\boldsymbol{N}(hk)\boldsymbol{x} > 0\}, & k < k_{\max}, \\ \mathbb{R}^{n_\zeta}, & k = k_{\max}, \end{cases} \tag{8}$$

$$\boldsymbol{N}(hk) := \begin{bmatrix} \boldsymbol{M}(hk) \\ \mathbf{I} \end{bmatrix}^{\mathsf{T}} \boldsymbol{Q} \begin{bmatrix} \boldsymbol{M}(hk) \\ \mathbf{I} \end{bmatrix}.$$

In (8), the set Q_k is the set of all points in the state-space that certainly triggered by time hk; then the set $\mathcal{R}_k$ is the isochronous subset obtained by removing from Q_k all points that have triggered before, i.e., $Q_j, j < k$. The sets $\mathcal{R}_k$ can be checked for non-emptiness exactly via nonlinear satisfiability-modulo-theories (SMT) solvers, e.g., Z3 [17], or approximately through a semi-definite relaxation (SDR) as proposed in [13]. The non-empty sets form the finite state-space of a *quotient system* [40], which gives formally a (exact) simulation relation to the original PETC traffic. Effectively, the sets $\mathcal{R}_k$ form a partition of $\mathbb{R}^{n_\zeta}$, and each $\mathcal{R}_k$ is a conjunction of finitely many quadratic cones, as depicted in Fig. 3.

The transition relation is as well a problem of checking non-emptiness of an intersection of quadratic inequalities:

$$\exists \boldsymbol{x} \in \mathcal{R}_i : \boldsymbol{M}(hk)\boldsymbol{x} \in \mathcal{R}_j \implies \mathcal{R}_i \xrightarrow{k} \mathcal{R}_j. \tag{9}$$

Similar to the CETC case, the state space of the abstraction is composed of the regions $\mathcal{R}_k$; however, the output set is now finite,

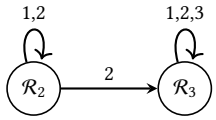


Figure 4: A simple PETC traffic model with only two regions and $k_{\max} = 3$..

$\mathcal{Y} = \{1, 2, ..., k_{\max}\}$. The transition set is composed of all triplets $\mathcal{R}_i \xrightarrow{k} \mathcal{R}_j$ satisfying (9); for scheduler synthesis (see Sec. 3.3), the action set is $\mathcal{U} = \mathcal{Y}$, meaning that the scheduler chooses the inter-sample time kh, often limited to be smaller than the PETC's inter-sample time ih. For verification, we drop the action set and the transitions are $\mathcal{R}_i \to \mathcal{R}_j := \mathcal{R}_i \xrightarrow{i} \mathcal{R}_j$. In both cases, we have effectively a finite transition system (see an example in Fig. 4), in contrast to the CETC case where the inter-sample time can still be chosen on a real interval.

Note that using relaxations for an existence problem such as the one above can lead to spurious transitions. From an abstraction (and simulation) perspective, this is acceptable, as it only adds nondeterminism to the abstraction; any scheduler that works for such an abstraction will work for the original system. Nevertheless, the more one can remove spurious non-determinism, the better, thus using exact solvers such as Z3 is recommended if the dimensions of the system permit. Further tightening can be obtained by using the refinements described in [15], where isochronous subsets were extended to *isosequential subsets*, that is, sets $\mathcal{R}_{k_1 k_2 k_3 \ldots k_l}$ containing the states whose next l inter-sample times are $hk_1, hk_2, \ldots hk_l$. This effectivelly chooses the *depth* of the abstraction refinement one wants to use for the traffic model. In ETCetera, the user can set the `depth` value and choose `solver in {'sdr', 'z3' }`; see §4.5 for an example.

3.3 Scheduler synthesis

Once one has constructed traffic models, they can be employed for the synthesis of schedulers. The goal of a scheduler, in our context, is to specify when each of the considered ETC systems should trigger, such that no overlap of their events occur, while still retaining stability of each of the ETC systems (which is guaranteed if always triggering before $\tau(x)$).

The tool currently considers two methods to synthesize schedulers. The first method relies on the use of UPPAAL. As already indicated in Section 3.1, the general traffic models constructed by ETCetera are equivalent to Timed Automata. Our tool exports the resulting models as timed automata that can be read and manipulated by UPPAAL to synthesize schedulers or check properties of the systems at hand.

The second synthesis method [42] follows from recognizing that the specialized traffic models constructed for PETC are finite transition systems (see Sec. 3.2). As such this method is only applicable for these traffic models. Furthermore it is assumed that the sampling time for each of the systems is identical. First, the representation of the traffic models is modified such that it acts on a per-sample basis. Instead of having actions corresponding to the next inter-sample time $\mathcal{U} = \mathcal{Y} = \{1, 2, \ldots, k_{max}\}$ as in the models described

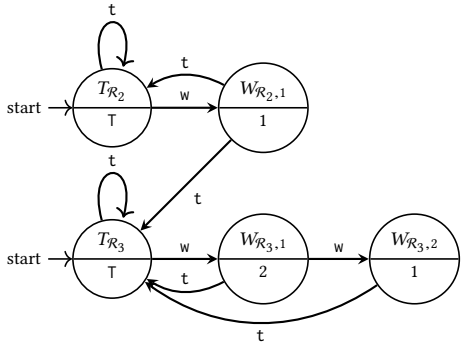


Figure 5: An example of wait-trigger model $\hat{\mathcal{S}}$ for PETC traffic.

in Section 3.2, a system can either wait ('w') , or trigger ('t') at the current sampling time. The result is a new transition system $\hat{\mathcal{S}} = (\hat{\mathcal{X}}, \hat{\mathcal{X}}_0, \hat{\mathcal{U}}, \hat{\mathcal{E}}, \hat{\mathcal{Y}}, \hat{H})$, where:

- $\hat{\mathcal{X}} := \{T_x \mid x \in \mathcal{X}\} \cup \{W_{x,j} \mid j < H(x),\, x \in \mathcal{X}\}$,
- $\hat{\mathcal{X}}_0 \subseteq \{T_x \mid x \in \mathcal{X}\}$,
- $\hat{\mathcal{U}} := \{w, t\}$,
- $\hat{\mathcal{E}} := \{(T_x, t, T_{x'}) \mid (x, 1, x') \in \mathcal{E}\} \cup \{(W_{x,j}, w, W_{x,j+1}) \mid 1 \leq j < H(x) - 1\} \cup \{(W_{x,j}, t, T_{x'}) \mid (x, j+1, x') \in \mathcal{E}\}$,
- $\hat{\mathcal{Y}} = \{T_1, T, 1, \ldots, k_{max} - 1\}$,
- $\hat{H}(T_x) = T_1$ if $H(x) = 1$, or T otherwise, and $\hat{H}(W_{i,j}) = H(i) - j$.

The outputs indicate either whether the system has just triggered (outputs T_1 and T)) or the triggering deadline. Some states are given, for technical reasons, the distinct output T_1 to capture that from those states the only action that can be taken is 't', while on the other states with output T_i ($i > 1$), both 'w' and 't' actions are enabled. In this transition system, triggering after k samples becomes a sequence of $k - 1$ 'w' actions followed by a single 't' action.

Next, all these transition systems (representing different PETC systems) are combined into a larger transition system, through a *parallel composition* allowing each subsystem to behave independently of the rest. Formally, the resulting composition is given by the transition system $\mathcal{S}_{comp.} = (\mathcal{X}_{comp.}, \mathcal{X}_{comp.;0}, \mathcal{U}_{comp.}, \mathcal{E}_{comp.}, \mathcal{Y}_{comp.}, H_{comp.})$, where

- $\mathcal{X}_{comp.} := \hat{\mathcal{X}}_0 \times \cdots \times \hat{\mathcal{X}}_n$,
- $\mathcal{X}_{comp.;0} := \hat{\mathcal{X}}_{0;0} \times \cdots \times \hat{\mathcal{X}}_{n;0}$,
- $\mathcal{U}_{comp.} := \hat{\mathcal{U}}_0 \times \cdots \times \hat{\mathcal{U}}_n$,
- $\mathcal{E}_{comp.} := \{((x_0, \ldots, x_n), (u_0, \ldots, u_n), (x'_0, \ldots, x'_n) \mid \forall i \leq n, (x_i, u_i, x'_i) \in \hat{\mathcal{E}}_i\}$,
- $\mathcal{Y}_{comp.} := \hat{\mathcal{Y}}_0 \times \cdots \times \hat{\mathcal{Y}}_n$,
- $H_{comp.}(x_0, \ldots, x_n) := (\hat{H}_0(x_0), \ldots, \hat{H}_n(x_n))$.

In this system the behaviour that a scheduler should prevent, i.e. two or more systems triggering simultaneously, is captured by actions which contain two or more more 't's. Here it is assumed that the fundamental checking period h of each of the PETC loops is the same. Since after triggering a subsystem always lands in a state with output T/T_1, the scheduler requirement can alternatively be expressed as: '*Avoid states in $\mathcal{S}_{comp.}$ whose output contains more than one T*'. Creating a controller that is able to avoid these states can be done effectively for finite transition systems by solving a safety game (c.f. Chapter 6 of [41]). In this safety game, the maximal fixed point of the operator

$$F_W(Z) = \{x \in Z \mid x \in W \wedge \exists u \in U(x) :\ \emptyset \neq Post_u(x) \subseteq Z\}, \quad (10)$$

is found by iterating over its output (starting with $Z_0 := \mathcal{X}_{comp.}$). The set W is the *safe set* which in this case is given by:

$$W = \{(x_1, \ldots, x_n) \in \mathcal{X}_{comp.} \mid |\{i \mid \hat{H}_i(x_i) \in \{T_1, T\}\}| \leq 1\}. \quad (11)$$

The maximal fixed point $Z^* := F_W(Z^*)$ contains states in $\mathcal{S}_{comp.}$ (given that $Z^* \neq \emptyset$) from which there exists at least one action guaranteeing that a bad state (a collision of triggering events) can be avoided. The scheduler can subsequently be defined as

$$U_c(x) = \{u \in \mathcal{U}_{comp.} \mid \emptyset \neq Post_u(x) \subseteq Z\}. \quad (12)$$

This is a function that returns a set of *safe* actions given some state $x \in \mathcal{X}_{comp.}$, which one can obtain from the last sampled states of each of the control systems. The scheduler keeps track of the current state of each $\hat{\mathcal{S}}_i$ and based on this collection of states, $U_c(x)$ provides a safe action, after which each of the states is updated according to $\hat{\mathcal{E}}_i$. Note that after a transmission, due to the nondeterminism of the original traffic model, one needs to rely on the value of the measured plant state to determine the corresponding successor state.

To improve the efficiency and scalability of this approach, two additional techniques are implemented (for details see [42]):

(1) *Partitioning and Refinement*: In this approach the size of each $\hat{\mathcal{S}}_i$ is reduced by grouping together similar states (*partitioning*). If no scheduler is found for these reduced systems, these blocks are split apart (*refinement*) and the safety game is performed again.
(2) *Binary Decision Diagrams (BDDs)* [6]: representing the transition systems with BDDs the manipulation of the transition systems (including partitioning and refinement) can be performed more efficiently, enabling the implementation of the safety-game fixed-point iteration symbolically.

3.4 Other synthesis and analysis problems

As mentioned in the introduction, traffic abstractions for PETC can also be used for quantitative analysis and other synthesis problems, as recently demonstrated in [12, 14]. For quantitative problems, the transition systems from Def. 1 are equipped with a *weight function* $\gamma : \mathcal{E} \to \mathbb{Q}$, mapping edges to rational numbers representing the *weight* or *value* of a transition. These systems are called *weighted transition systems* (WTSs) [7]. For every run $x_0 \xrightarrow{u_0} x_1 \xrightarrow{u_1} x_2 \ldots$ of $\mathcal{S}$ we can thus compute a sequence of weights $w_0 w_1 w_2 \ldots$, giving rise to the *set of all weight sequences* $\mathcal{W}_{\mathcal{S}}$. For ETC, we are interested in computing or maximizing the *smallest average inter-sample time* (SAIST) of a system; hence, the weight of a transition is its inter-sample time. The SAIST of a PETC traffic model $\mathcal{S}$ is formally defined as

$$\text{SAIST}(\mathcal{S}) = \inf_{w_0 w_1 w_2 \ldots \in \mathcal{W}_{\mathcal{S}}} \liminf_{n \to \infty} \frac{1}{n+1} \sum_{i=0}^{n} w_i. \quad (13)$$

A lower bound to the SAIST of a PETC system can be computed using Karp's algorithm [8, 31] on the abstraction. In some cases, the

abstraction can be a *smallest-average-cycle-equivalent* simulation [14, 16], a property that can be verified for PETC of linear systems: in this case, the SAIST from the abstraction is exact.

In order to do synthesis, we equip the PETC traffic model with early-triggering samples as in the scheduling problem, and solve a *mean-payoff game* [25], obtaining a lower bound to the optimal sampling strategy [12]. The strategy is a function mapping a region $\mathcal{R}_i$ onto an inter-sample time choice k, becoming effectively a self-triggered control scheme [4, 34, 44]. The idea behind the approach in [12] is to sample early in some regions of the state-space to collect long-term benefits, something that the shortsighted ETC cannot do. The abstractions contain long-term traffic information which allows better strategies to be computed. In [12], the SAIST of a numerical example nearly doubled using this approach.

3.5 Discussion on computational complexity

When abstracting general CETC systems, the size of the state-space partition using either gridding or IMs scales exponentially with n_ζ. Nonetheless, in the case of linear PETC, the use of isochronous sets removes the dependence on dimensions, and the number of regions is $O(k_{max}^l)$. The complexity of computing IM-approximations mainly depends on the complexity of the dReal [27], which is exponential on n_ζ for our problem. Generating timing intervals and transitions, in the case of general CETC systems, is $O(r)C$ and $O(r^2)C$ respectively, where r is the number of regions and C is the complexity of the reachability analysis operation. For linear PETC, the transition relation computation is $O(k_{max}r^2)C'$, where C' is polynomial in n_ζ when using `solver='sdr'` (see [13]) and exponential in n_ζ when using `solver='z3'` (see [16]). The resulting system $\hat{S}$ for linear PETC has $O(k_{max}^{l+1})$ states (which is rk_{max}) and $O(k_{max}^{2l+2})$ transitions (which is rk_{max}^2). In practice, the number of regions often grows much slower than exponentially with l, a phenomenon investigated in [28]. Regarding scheduling, using UPPAAL the size of the problem scales exponentially with the number of systems in the network (TA safety games are exponential on the number of clocks). The scheduling technique of Section 3.3 is more efficient, exploiting PETC models being already FTSs. Additionally, partitioning can reduce the number of states to $O(k_{max})$ (and the transitions accordingly to $O(k_{max}^2)$). However, if the safety game fails, refinement has to be performed, increasing the number of states. Introducing BDDs complicates this analysis, see [42] for more details. It is important to remark, however, that the abstraction and scheduler synthesis processes are performed offline, with the synthesized scheduler's online implementation being simply a set of if-then-else rules.

4 USE OF ETCETERA

ETCetera is implemented in Python, and uses third-party tools cvxpy [2, 24], z3 [17], dReach [33], dReal [27] and Flowstar [9] for the creation of traffic models, UPPAAL Stratego [11] for scheduler synthesis and dd [10] (with bindings to CUDD [38]) for the manipulation of BDDs. To construct traffic models, the user has two possibilities: (i) employ a command line interface, or (ii) employ the utilities provided inside a python script, as illustrated in detail in the examples in Sections 4.2 and 4.3. Analysis and scheduler synthesis can currently only be performed within a Python script, as shown in the examples of Sections 4.4 and 4.5. The computations of all examples below have been conducted on a laptop with an Intel i7-9750H processor and 16 GB of RAM.

4.1 Command line interface

If the command line interface is employed, the user simply needs to run the following command:

```
$ python etc2traffic.py system_type input_file [options]
```

This reads the contents of *input_file* and generates a traffic model. The contents of *input_file* depend on the *system_type* (either *linear PETC* or *general*). When *system_type* is *linear PETC*, *input_file* looks like:

```
Dynamics : [0 1; -2 3], [0; 1]
Controller: [1 -4]
Triggering Sampling Time: 0.01
Triggering Heartbeat: 0.40
Triggering Condition: [0.95 0 -1 0;0 0.95 0 -1;-1 0 1 0;0
    ↪  -1 0 1]
```

This constructs a specialized PETC traffic model as is described in Section 3.2. *Dynamics* contains the state evolution matrix ($\boldsymbol{A}$) and the input matrix ($\boldsymbol{B}$), and the *Controller* contains the feedback gain ($\boldsymbol{K}$). These must all be expressed as matrices. *Triggering Sampling Time* is the checking period h that is considered and *Triggering Heartbeat* is the maximum trigger time (hk_{max}). Finally, *Triggering Condition* contains the triggering matrix $\boldsymbol{Q}$ as only quadratic triggering conditions are considered in this case.

When *system_type* is *general*, *input_file* looks like:

```
Hyperbox States : [-2 2], [-2 2]
Grid Points Per Dimension: [3 3]
Dynamics : x1, x1**2*x2 + x2**3 + u1
Controller: -x2 - x1**2*x2 - x2**3
Triggering Condition : e1**2 + e2**2 - 0.01**2
Solver Options : manifolds_times=[0.002 0.0028 0.0038
    ↪ 0.005 0.0065 0.0075], partition_method=manifold,
    ↪ heartbeat=0.021, order_approx=4
```

Which constructs a general traffic model as described in Section 3.1. In this case, *Dynamics*, *Controller* and *Triggering Conditions* are specified as symbolic expressions (where ** represents exponentiation). State variables are denoted with `x1, x2, ...`, control variables with `u1, u2, ...`, error variables with `e1, e2, ...`, disturbance variables with `d1, d2, ...` and the variable used for homogenization as `w1`. Furthermore, the rectangular region of the state space that is to be considered must be supplied in *Hyperbox States*. Other mandatory arguments depend on: (i) the partition method, (ii) whether the system is homogeneous, (iii) if disturbances are involved:

- If `partition_method=grid` or when the system is not homogeneous, `Grid Points Per Dimension` must be specified, which contains the number of regions the state space is divided into for each of the states.
- If `partition_method=manifold`, then `manifolds_times` in `Solver Options` must be specified, which will define the Isochronous Manifolds.
- If disturbances are present, `Hyperbox Disturbances` must be specified, containing for each disturbance the interval it will lie in.

Other optional parameters can be supplied in `Solver Options`. In this example `heartbeat` contains the maximum trigger time and `order_approx` contains the order of approximation of the isochronous manifolds.

In addition to this command line interface, one can construct traffic models via Python scripts as well, as will be shown in Sections 4.2 and 4.3. The command line interface is currently only available for the construction of traffic models, so for scheduling and analysis purposes the procedures in Sections 4.4 and 4.5 should be followed.

4.2 Python interface: linear PETC abstractions

Consider the linear PETC system:

$$\begin{aligned}\dot{\zeta}(t) &= \begin{bmatrix}-0.5 & 0\\ 0 & 3.5\end{bmatrix}\zeta(t) + \begin{bmatrix}1\\1\end{bmatrix}u(\zeta(t_i)),\\ u(\zeta(t_i)) &= \begin{bmatrix}1.02 & -5.62\end{bmatrix}\zeta(t_i),\end{aligned} \qquad t\in[t_i,t_{i+1}) \tag{14}$$

with triggering function $\phi(\zeta(t),\zeta(t_i)) = |\zeta(t)-\zeta(t_i)|^2-(1-\sigma)\,|\zeta(t)|^2$, where $\sigma = 0.05$ and $h = 0.01$, $k_{max} = 20$. The PETC traffic model for this system is generated as follows:

```
import numpy as np

# Define LTI system matrices
A = np.array([[-0.5, 0], [0, 3.5]])
B = np.array([[1],[1]])
K = np.array([[1.02, -5.62]])

# PETC parameters
h = 0.01
kmax = 20
sigma = 0.05

# Triggering condition
Qtrigger = np.block([[(1-sigma)*np.eye(2), -np.eye(2)], [
    ↪ -np.eye(2), np.eye(2)]])

# Construct object representing the PETC system
import ETCetera.Systems as systems

plant = systems.LinearPlant(A, B)
controller = systems.LinearController(K, h)
trigger = systems.LinearQuadraticPETC(plant, controller,
    ↪ kmax=kmax, Qbar=Qtrigger)

# Create Traffic Model
import ETCetera.Abstractions as abstr

traffic = abstr.TrafficModelLinearPETC(trigger)
regions, transitions = traffic.create_abstraction()
```

4.3 Python interface: nonlinear CETC abstractions

Consider the nonlinear CETC system:

$$\begin{aligned}\dot{\zeta}(t) &= \begin{bmatrix}\zeta_1(t)\\ \zeta_1^2(t)\zeta_2(t)+\zeta_2^3(t)+u(\zeta(t_i))\end{bmatrix},\\ u(\zeta(t_i)) &= -\zeta_2(t_i)-\zeta_1^2(t)\zeta_2(t)-\zeta_2^3(t),\end{aligned} \qquad t\in[t_i,t_{i+1}) \tag{15}$$

together with the triggering condition $\phi(\zeta(t),\zeta(t_i)) = |\zeta(t)-\zeta(t_i)|^2-0.01^2$. The CETC system may be defined as follows:

```
import sympy

# Define
state_vector = x1, x2, e1, e2 \
    = sympy.symbols('x1 x2 e1 e2')

# Define controller (in etc form)
u1 = -(x2+e2) - (x1+e1)**2*(x2+e2) - (x2+e2)**3

# Define dynamics
x1dot = -x1
x2dot = x1**2*x2 + x2**3 + u1
dynamics = [x1dot, x2dot, -x1dot, -x2dot]

# Triggering condition & other etc.
trigger = e1 ** 2 + e2 ** 2 - 0.01**2

# State space limits
state_space_limits = [[-2, 2], [-2, 2]]
grid_points_per_dim = [3,3]
```

A traffic model for this system is then simply constructed with:

```
import ETCetera.Abstractions as abstr

# Partition method manifold
traffic = abstr.TrafficModelNonlinearETC(dynamics,
    ↪ trigger, state_vector, state_space_limits=
    ↪ state_space_limits, grid_points_per_dim=
    ↪ grid_points_per_dim, manifolds_times=[0.002,
    ↪ 0.0028, 0.0038, 0.005, 0.0065, 0.0075],
    ↪ partition_method='manifold', heartbeat=0.021,
    ↪ order_approx=4)
regions, transitions = traffic.create_abstraction()
```

Plots of lower and upper bounds on the inter-sample times and transitions for the resulting traffic model can be generated by running the command `traffic.visualize()`, and they are shown in Figure 6. The total CPU time was about 1 hour.

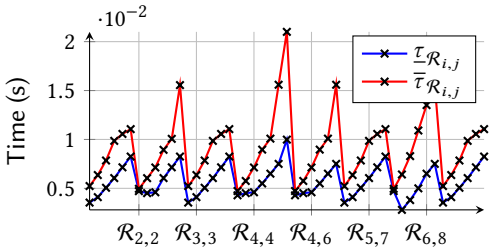
To add disturbances to the dynamics, the corresponding parameters (`d1,...`) and their domain `dist_param_domain` have to be declared, as well as `partition_method='grid'` has to be specified:

```
d1 = sympy.symbols('d1')
x1dot = -x1
x2dot = x1**2*x2 + x2**3 + u1 + d1
dynamics = [x1dot, x2dot, -x1dot, -x2dot]
disturbace_limits = [[-0.1, 0.1]]
traffic = abstr.TrafficModelNonlinearETC(dynamics,
    ↪ trigger, state_vector, state_space_limits=
    ↪ state_space_limits, dist_param=(d1,),
    ↪ dist_param_domain=disturbace_limits, order_approx
    ↪ =2, heartbeat=0.022, grid_points_per_dim=[7,8],
    ↪ partition_method='grid', precision_deltas=1e-6)
```

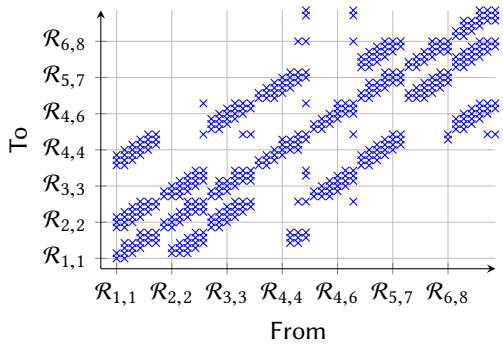
4.4 Scheduler synthesis

In this example, the complete workflow of generating a scheduler for two linear PETC systems is described. Consider the two systems [30, 40]:

$$\begin{aligned}\dot{\zeta}_1(t) &= \begin{bmatrix}0 & 1\\ -2 & 3\end{bmatrix}\zeta_1(t) + \begin{bmatrix}0\\1\end{bmatrix}u_1(\zeta_1(t_i)),\\ u_1(\zeta_1(t_i)) &= \begin{bmatrix}1 & -4\end{bmatrix}\zeta_1(t_i),\\ \dot{\zeta}_2(t) &= \begin{bmatrix}-0.5 & 0\\ 0 & 3.5\end{bmatrix}\zeta_2(t) + \begin{bmatrix}1\\1\end{bmatrix}u_2(\zeta_2(t_i)),\\ u_2(\zeta_2(t_i)) &= \begin{bmatrix}1.02 & -5.62\end{bmatrix}\zeta_2(t_i),\end{aligned} \qquad t\in[t_i,t_{i+1}) \tag{16}$$

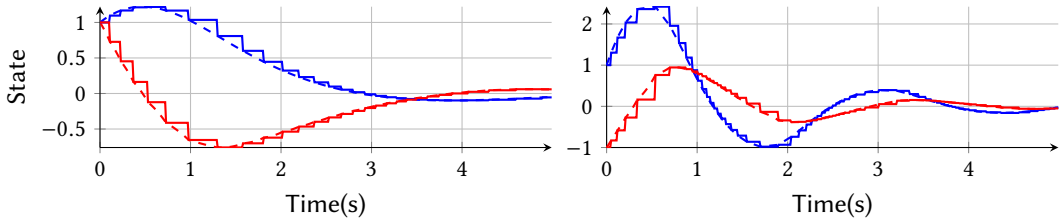


(a) Lower and upper bounds of the inter-sample times for each of the regions of the traffic model generated in Example 4.3



(b) Transition between the regions of the traffic model generated in Example 4.3

Figure 6: Example 4.3 results.



(a) State evolution of system 1 with initial condition $x_0 = [1, 1]^T$.(b) State evolution of system 2 with initial condition $x_0 = [1, -1]^T$.

Figure 7: Evolution of the systems' states under the scheduler synthesized as detailed in Section 4.4. Dashed lines represent the actual state evolution, while the solid lines represent the states available to the controller at each time.

both with triggering condition $\phi(\zeta(t), \zeta(t_i)) = |\zeta(t) - \zeta(t_i)|^2 - (1 - \sigma)|\zeta(t)|^2$, where $\sigma = 0.05$ and $h = 0.01$, $k_{1,max} = 40$, $k_{2,max} = 20$. These two systems have been defined in files as specified in Section 4.1. A scheduler can then be synthesized and simulated as follows:

```
from ETCetera.util import *

# First construct the traffic models
traffic1 = construct_linearPETC_traffic_from_file('
    ↪ examples/linear_2.txt')
traffic2 = construct_linearPETC_traffic_from_file('
    ↪ examples/linear_3.txt')

import ETCetera.Scheduling.fpiter as sched

# Convert traffic models into different form
cl1 = sched.controlloop(traffic1)
cl2 = sched.controlloop(traffic2)

# Construct system and generate scheduler
S = sched.system([cl1, cl2])
Ux, _ = S.generate_safety_scheduler()

# Simulate
S.simulate(x0=[[1,1],[1,-1]], Tmax=5)
```

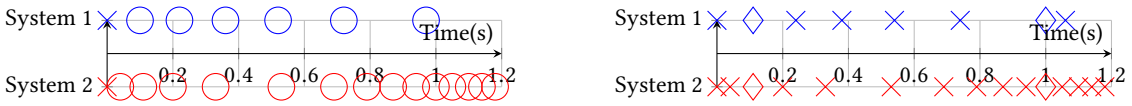
The results of the simulation are shown in Figures 7 and 8. The total CPU time was 10 minutes. By default, this synthesis approach makes use of BDDs. To make this process more insightful, one can specify `use_bdd=False` in the construction of the control loop. However, this can drastically impact performance of the synthesis process.

Creating schedulers using UPPAAL follows the same steps, with the exception that additionally a network has to be defined:

```
import ETCetera.Scheduling.NTGA as sched

cl1 = sched.controlloop(traffic1)
cl2 = sched.controlloop(traffic2)

# Define a network and combine
net = sched.network()
nta = sched.nta(net, [cl1, cl2])

# Generate a strategy using UPPAAL and parse its results
nta.generate_strategy(parse_strategy=True)
```

4.5 Quantitative analysis and synthesis

Consider the first system in (16). In this example we are interested in obtaining a large average inter-sample time; hence we use the



(a) Sampling times with the use of the synthesized scheduler.

(b) Sampling times without the use of a scheduler.

Figure 8: Sampling times of the first 1.2 seconds of the simulation in Figure 7. For reference, the sampling times when no scheduler is used are shown in (b). Early triggers are denoted with a '○'. Sampling collisions are represented with a '◇'.

more aggressive Lyapunov-based triggering condition [15, 39] of the form $V(\hat{\boldsymbol{\zeta}}(t)) > -\rho\hat{\boldsymbol{\zeta}}(t)^{\mathsf{T}}\boldsymbol{Q}_{\text{Lyap}}\hat{\boldsymbol{\zeta}}(t)$, where $\hat{\boldsymbol{\zeta}}(t) \coloneqq \boldsymbol{A}_{\text{d}}(h)\boldsymbol{\zeta}(t) + \boldsymbol{B}_{\text{d}}(h)\boldsymbol{K}\boldsymbol{\zeta}(t_i)$ is the state prediction after h time units, $V(\boldsymbol{x}) = \boldsymbol{x}^{\mathsf{T}}\boldsymbol{P}_{\text{Lyap}}\boldsymbol{x}$, and $0 < \rho < 1$ is the triggering parameter. For the numerical results below, the Lyapunov matrices were taken from [40] as $\boldsymbol{P}_{\text{Lyap}} = \left[\begin{smallmatrix} 1 & 0.25 \\ 0.25 & 1 \end{smallmatrix}\right]$, $\boldsymbol{Q}_{\text{Lyap}} = \left[\begin{smallmatrix} 0.5 & 0.25 \\ 0.25 & 1.5 \end{smallmatrix}\right]$, and we set $h = 0.1$, $\rho = 0.8$ and $K = 20$. The triggering condition used is quadratic and can be specified as in §4.2. The code snippet below (replacing lines 26–27 from the example in §4.2) shows how to

(1) construct abstractions to compute the system's smallest average inter-sample time (SAIST), where we flag `etc_only=True` to avoid computing unnecessary early-triggering transitions;
(2) construct a normal abstraction with early-triggering transitions to synthesize a near-optimal sampling strategy;
(3) verify the SAIST of the closed-loop system with the generated strategy.

```
26 # Best selection for SAIST: solver='z3', etc_only,
       ↪ stop_if_mace_equivalent, smart_mace
27 traffic = abstr.TrafficModelLinearPETC(trigger, depth
       ↪ =100, etc_only=True, solver='z3',
       ↪ stop_if_mace_equivalent=True, smart_mace=True)
28 regions, transitions = traffic.create_abstraction()
29
30 # Print obtained SAIST and its minimizing cycle
31 print(f'SAIST is {traffic.limavg}'); print(f'Smallest
       ↪ average cycles: {traffic.cycles}')
32
33 # Now create an abstraction to design an improved
       ↪ sampling strategy
34 traffic_synth = abstr.TrafficModelLinearPETC(trigger,
       ↪ depth=1, etc_only=False, solver='z3',
       ↪ early_trigger_only=True)
35 strat, strat_tran = traffic_synth.
       ↪ optimize_sampling_strategy()
36 print(strat) # Inspect strategy
37
38 # Compute its SAIST value
39 traffic2 = abstr.TrafficModelLinearPETC(trigger, depth
       ↪ =100, etc_only=True, solver='z3',
       ↪ stop_if_mace_equivalent=True, smart_mace=True,
       ↪ strategy=strat, strategy_transition=strat_tran)
40 regions2, transitions2 = traffic2.create_abstraction()
41 print(f'Optimized SAIST is {traffic2.limavg}')
```

This code reproduces part of the results in [12], giving the following output (the numbers are normalized by h):

```
SAIST is 2.3333333333333335
Smallest average cycles: {(8, 1, 1, 1, 1, 2)}
Optimized SAIST is 5.0
```

The algorithm terminated in 5 minutes.

5 CONCLUSIONS AND FUTURE WORK

We have introduced the tool ETCetera enabling the study and manipulation of the inter-sample times generated by event-triggered control systems. The main functionality of the tool is to create abstract *traffic models* from the inter-sample times dynamics implicitly determined by the hybrid dynamics of ETC systems. The tool allows to construct traffic models in the form of timed-automata for general ETC systems, and in the form of finite transition systems in the particular case of PETC systems. The resulting models can be exported for further analysis to UPPAAL, in the case of timed-automata models, or manipulated within the tool (in the case of PETC systems). ETCetera provides algorithmic implementations to synthesize schedulers, compute performance metrics, and design improved sampling strategies for traffic models of PETC systems. The tool is available as an open-source project and continuously being extended with additional functionalities. In particular ongoing and future work is focusing on the following:

- IM-based partitioning for systems with disturbances. Even though the theory developed in [21] addresses this case, this is not yet implemented as Flow* cannot handle ball-segments and dReach cannot handle disturbances.
- In [21] it has been observed that IM-based partitioning, sometimes, results into more transitions, probably because the sets $\mathcal{R}_i$ are approximated by ball-segments (which may be a crude approximation). We are working on tackling this issue (with tighter approximations) in future versions of the tool.
- Based on the developments of [18] and [19], extend abstractions to stochastic PETC systems via *interval markov decision processes*, for verification of various performance metrics and for scheduler synthesis with probabilistic safety.
- Automate the synthesis of schedulers for PETC loops not sharing the same fundamental sampling time h.
- Extend the specialized models of linear PETC traffic to perturbed systems, output-feedback systems and CETC.
- Enable the simulation of multiple ETC loops under a synthesized scheduler over the network simulator OMNET++ [43].

ACKNOWLEDGMENTS

The authors thank Dr. Khushraj Madnani for developing the abstraction minimization code and his overall support with scheduler synthesis algorithms, Dr. Cees F. Verdier for assisting in developing the part of the code which generates the IMs, and Dr. Gururaj Maddodi for assisting in the overall architecture of the tool's code. This work is supported by the European Research Council through the SENTIENT project, Grant No. ERC-2017-STG #755953 (https://cordis.europa.eu/project/id/755953).

REFERENCES

[1] Karl-Erik Åarzén. 1999. A simple event-based PID controller. *IFAC Proceedings Volumes* 32, 2 (1999), 8687–8692.

[2] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. 2018. A rewriting system for convex optimization problems. *Journal of Control and Decision* 5, 1 (2018), 42–60.

[3] Rajeev Alur and David L Dill. 1994. A theory of timed automata. *Theoretical computer science* 126, 2 (1994), 183–235.

[4] Adolfo Anta and Paulo Tabuada. 2008. Self-triggered stabilization of homogeneous control systems. In *American Control Conference, 2008.* IEEE, 4129–4134.

[5] Karl Johan Astrom and Bo M Bernhardsson. 2002. Comparison of Riemann and Lebesgue sampling for first order stochastic systems. In *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, Vol. 2. IEEE, 2011–2016.

[6] Randal E Bryant. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)* 24, 3 (1992), 293–318.

[7] Krishnendu Chatterjee, Laurent Doyen, and Thomas A Henzinger. 2010. Quantitative languages. *ACM Transactions on Computational Logic (TOCL)* 11, 4 (2010), 1–38.

[8] Mmanu Chaturvedi and Ross M McConnell. 2017. A note on finding minimum mean cycle. *Inform. Process. Lett.* 127 (2017), 21–22.

[9] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. 2013. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*. Springer, 258–263.

[10] Caltech Control and Dynamical Systems. 2020. dd (version 0.5.6.). https://pypi.org/project/dd/.

[11] Alexandre David, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marius Mikučionis, and Jakob Haahr Taankvist. 2015. Uppaal stratego. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 206–211.

[12] Gabriel de A. Gleizer, Khushraj Madnani, and Manuel Mazo Jr. 2021. Self-Triggered Control for Near-Maximal Average Inter-Sample Time. In *60th IEEE Conference on Decision and Control (accepted)*.

[13] Gabriel de A. Gleizer and Manuel Mazo Jr. 2020. Scalable traffic models for scheduling of linear periodic event-triggered controllers. *IFAC-PapersOnLine* 53, 2 (2020), 2726–2732.

[14] Gabriel de A. Gleizer and M. Mazo Jr. 2021. Computing the sampling performance of event-triggered control. In *Proc. of the 24th Int'l Conf. on Hybrid Systems: Computation and Control* (Nashville, TN, USA) *(HSCC '21)*. ACM, Article 20, 7 pages.

[15] Gabriel de A. Gleizer and Manuel Mazo Jr. 2021. Towards Traffic Bisimulation of Linear Periodic Event-Triggered Controllers. *IEEE Control Systems Letters* 5, 1 (2021), 25–30.

[16] Gabriel de Albuquerque Gleizer and Manuel Mazo Jr. 2021. Computing the average inter-sample time of event-triggered control using quantitative automata. arXiv:2109.14391 [eess.SY]

[17] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.

[18] Giannis Delimpaltadakis, Luca Laurenti, and Manuel Mazo Jr. 2021. Abstracting the Sampling Behaviour of Stochastic Linear Periodic Event-Triggered Control Systems. In *2021 60th IEEE Conference on Decision and Control (CDC)*. 1287–1294. https://doi.org/10.1109/CDC45484.2021.9683751

[19] Giannis Delimpaltadakis, Luca Laurenti, and Manuel Mazo Jr. 2022. Formal Analysis of the Sampling Behaviour of Stochastic Event-Triggered Control. *arXiv preprint arXiv:2202.10178* (2022).

[20] Giannis Delimpaltadakis and Manuel Mazo Jr. 2020. Traffic Abstractions of Nonlinear Homogeneous Event-Triggered Control Systems. In *2020 59th IEEE Conference on Decision and Control (CDC)*. 4991–4998. https://doi.org/10.1109/CDC42340.2020.9303968

[21] Giannis Delimpaltadakis and Manuel Mazo Jr. 2021. Abstracting the Traffic of Nonlinear Event-Triggered Control Systems. *arXiv preprint arXiv:2010.12341, under review* (2021).

[22] Giannis Delimpaltadakis and Manuel Mazo Jr. 2021. Isochronous Partitions for Region-Based Self-Triggered Control. *IEEE Trans. Automat. Control* 66, 3 (2021), 1160–1173. https://doi.org/10.1109/TAC.2020.2994020

[23] Giannis Delimpaltadakis and Manuel Mazo Jr. 2021. Region-Based Self-Triggered Control for Perturbed and Uncertain Nonlinear Systems. *IEEE Transactions on Control of Network Systems* 8, 2 (2021), 757–768. https://doi.org/10.1109/TCNS.2021.3050121

[24] Steven Diamond and Stephen Boyd. 2016. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research* 17, 83 (2016), 1–5.

[25] Andrzej Ehrenfeucht and Jan Mycielski. 1979. Positional strategies for mean payoff games. *International Journal of Game Theory* 8, 2 (1979), 109–113.

[26] Christophe Fiter, Laurentiu Hetel, Wilfrid Perruquetti, and Jean-Pierre Richard. 2012. A state dependent sampling for linear state feedback. *Automatica* 48, 8 (2012), 1860–1867.

[27] Sicun Gao, Soonho Kong, and Edmund M Clarke. 2013. dReal: An SMT solver for nonlinear theories over the reals. In *International Conference on Automated Deduction*. Springer, 208–214.

[28] Gabriel de Albuquerque Gleizer and Manuel Mazo Jr. 2022. Chaos and order in event-triggered control. *arXiv preprint arXiv:2201.04462* (2022).

[29] W. P. M. H. Heemels, M. C. F. Donkers, and Andrew R. Teel. 2013. Periodic event-triggered control for linear systems. *IEEE Trans. Automat. Control* 58, 4 (2013), 847–861.

[30] L. Hetel, A. Kruszewski, W. Perruquetti, and J. Richard. 2011. Discrete and Intersample Analysis of Systems With Aperiodic Sampling. *IEEE Trans. Automat. Control* 56, 7 (2011), 1696–1701. https://doi.org/10.1109/TAC.2011.2122690

[31] Richard M Karp. 1978. A characterization of the minimum cycle mean in a digraph. *Discrete mathematics* 23, 3 (1978), 309–311.

[32] Arman Sharifi Kolarijani and Manuel Mazo Jr. 2016. Formal Traffic Characterization of LTI Event-Triggered Control Systems. *IEEE Transactions on Control of Network Systems* 5, 1 (2016), 274–283.

[33] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. 2015. dReach: δ-reachability analysis for hybrid systems. In *International Conference on TOOLS and Algorithms for the Construction and Analysis of Systems*. Springer, 200–205.

[34] Manuel Mazo Jr., Adolfo Anta, and Paulo Tabuada. 2010. An ISS self-triggered implementation of linear controllers. *Automatica* 46, 8 (2010), 1310–1314.

[35] Manuel Mazo Jr, Arman Sharifi-Kolarijani, Dieky Adzkiya, and Christiaan Hop. 2018. Abstracted models for scheduling of event-triggered control data traffic. In *Control Subject to Computational and Communication Constraints*. Springer, 197–217.

[36] R. Postoyan, R. G. Sanfelice, and W. P. M. H. Heemels. 2019. Inter-event Times Analysis for Planar Linear Event-triggered Controlled Systems. In *2019 IEEE 58th Conference on Decision and Control (CDC)*. 1662–1667. https://doi.org/10.1109/CDC40024.2019.9028888

[37] A. Rajan and P. Tallapragada. 2020. Analysis of Inter-Event Times for Planar Linear Systems Under a General Class of Event Triggering Rules. In *2020 59th IEEE Conference on Decision and Control (CDC)*. 5206–5211. https://doi.org/10.1109/CDC42340.2020.9304406

[38] F. Somenzi. 1998. CUDD: CU Decision Diagram Package Release 2.2.0.

[39] Aleksandra Szymanek, Gabriel de A. Gleizer, and Manuel Mazo Jr. 2019. Periodic event-triggered control with a relaxed triggering condition. In *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 1656–1661.

[40] Paulo Tabuada. 2007. Event-triggered real-time scheduling of stabilizing control tasks. *IEEE Trans. Automat. Control* 52, 9 (2007), 1680–1685.

[41] Paulo Tabuada. 2009. *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media.

[42] Ivo van Straalen. 2021. *Efficient Scheduler Synthesis For Periodic Event Triggered Control Systems: An Approach With Binary Decision Diagrams*. Master's thesis. Delft University of Technology. http://resolver.tudelft.nl/uuid:f9764019-e908-45e7-a053-0f6fbc8a7792

[43] Andras Varga. 2010. OMNeT++. In *Modeling and tools for network simulation*. Springer, 35–59.

[44] Manel Velasco, Josep Fuertes, and Pau Marti. 2003. The self triggered task model for real-time control systems. In *Work-in-Progress Session of the 24th IEEE Real-Time Systems Symposium (RTSS03)*, Vol. 384.