Offline and online energy-efficient monitoring of scattered uncertain logs using a bounding model

Bineet Ghosh, Étienne André

Logical Methods in Computer Science
Volume 20, Issue 1, 2024, pp. 2:1–2:33
https://lmcs.episciences.org/

Submitted Dec. 07, 2022
Published Jan. 11, 2024

OFFLINE AND ONLINE ENERGY-EFFICIENT MONITORING OF SCATTERED UNCERTAIN LOGS USING A BOUNDING MODEL

BINEET GHOSH [a,b] AND ÉTIENNE ANDRÉ [c,d]

[a] The University of Alabama, AL, The United States of America
e-mail address: bineet@ua.edu

[b] The University of North Carolina at Chapel Hill, NC, The United States of America

[c] Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

[d] Université Sorbonne Paris Nord, LIPN, CNRS UMR 7030, F-93430 Villetaneuse, France

Abstract. Monitoring the correctness of distributed cyber-physical systems is essential. Detecting possible safety violations can be hard when some samples are uncertain or missing. We monitor here black-box cyber-physical system, with logs being uncertain both in the state and timestamp dimensions: that is, not only the logged value is known with some uncertainty, but the time at which the log was made is uncertain too. In addition, we make use of an over-approximated yet expressive model, given by a non-linear extension of dynamical systems. Given an offline log, our approach is able to monitor the log against safety specifications with a limited number of false alarms. As a second contribution, we show that our approach can be used online to minimize the number of sample triggers, with the aim at energetic efficiency. We apply our approach to three benchmarks, an anesthesia model, an adaptive cruise controller and an aircraft orbiting system.

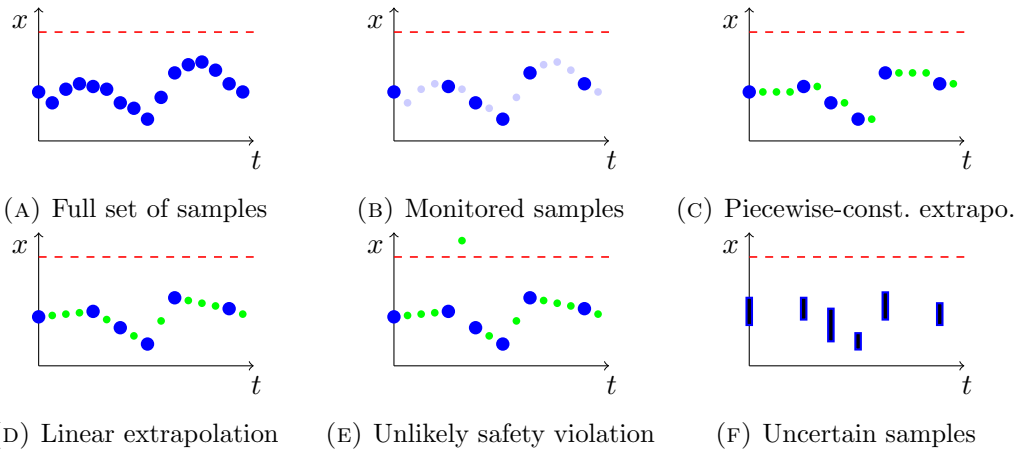
1. Introduction

The pervasiveness of distributed cyber-physical systems is highly increasing, accompanied by associated safety concerns. Formal verification techniques usually require a (white-box) model, which may not often available, because some components are black-box, or because the entire system has no formal model. In addition, despite some success in verifying formal models from the industry in the recent past (e.g., [BCM+92, KGN+09, LLN18, ACF+21]), formal verification techniques for cyber-physical systems are often subject to state space explosion, often preventing a satisfactory scalability (see e.g., [Pel08, CKNZ11]). Therefore, *monitoring*, as a lightweight yet feasible verification technique, can bring practical results of high importance for larger models.

Monitoring aims at analyzing the log of a concrete system, so as to deduce whether a specification (e.g., a safety property) is violated [BDD+18]. Monitoring can be done *offline* (i.e., after the system execution, assuming the knowledge of the entire log, see e.g.,

Key words and phrases: offline monitoring, online monitoring, energy-aware monitoring, cyber-physical systems, formal methods.

DOI:10.46298/LMCS-20(1:2)2024



(A) Full set of samples (B) Monitored samples (C) Piecewise-const. extrapo.

(D) Linear extrapolation (E) Unlikely safety violation (F) Uncertain samples

FIGURE 1. Monitoring at discrete time steps

[BCE+16]), or *online* (at runtime, assuming a partial log); see [Mal16] for a discussion on online verification. When the log is an aperiodic timed sequence of valuations of continuous variables, with a logging *not* occurring at every discrete time step, and when the system under monitoring is a black box, a major issue is: how to be certain that, in between two discrete valuations, the specification was not violated at another discrete time step at which no logging was performed? For example, consider a system for which a logging at every discrete time step would yield the log depicted in Fig. 1a. Assume the logging was done at only *some* time steps, given in Fig. 1b, due to some sensor faults, or to save energy with only a sparse, scattered logging. How to be certain that, in between two discrete samples, another discrete sample (not recorded) did not violate the specification? For example, by just looking at the discrete samples in Fig. 1b, there is no way to formally guarantee that the unsafe zone (i.e., above the red, dashed line) was never reached by another discrete sample which was not recorded. In many practical cases, a piecewise-constant or linear approximation (see, e.g., Figs. 1c and 1d, where the large blue dots denote actual samples, while the small green dots denote reconstructed samples using some extrapolation) is arbitrary and not appropriate; even worse, it can yield a "safe" answer, while the actual system could actually have been unsafe at some of the missing time steps. On the contrary, assuming a completely arbitrary dynamics will always yield "potentially unsafe"—thus removing the interest of monitoring. For example, from the samples in Fig. 1b, without any knowledge of the model, one can always envision the situation in Fig. 1e, which shows the variable x crossing the unsafe region (dashed) at some unlogged discrete time step—even though this is unlikely if the dynamics is known to vary "not very fast".

Contributions. In this work, we address the problem of performing monitoring over a set of scattered and *uncertain* samples. First, we cope with uncertainties from the sensors by allowing for *uncertain* samples, given by zonotopes over the continuous variables; that is, at each logged timestamp, the log gives not a constant value for the continuous variables, but a *zonotope*.[1] For example, let us examine the case of an adaptive cruise control (ACC). In ACC, it is crucial to accurately measure the distance to the vehicle in front for maintaining safety. Nonetheless, due to uncertainties in the sensors, it may not always be feasible to

[1]A zonotope is a special form of a convex polyhedron that is centrally symmetric.

obtain the precise distance of the lead vehicle. However, it may still be possible to measure this distance within a certain margin of error. Consequently, the measured distance of the lead vehicle becomes an interval of values that more accurately represents the true distance, taking into account the potential inaccuracies inherent in the sensor's error range. For instance, when the reading is taken using a sensor with a $\pm 5\%$ error and the value read by the sensor is 1, the uncertain value can be calculated by accounting for the sensor's error tolerance, resulting in a range of $[0.95, 1.05]$. In this work, to represent all state variable values along with their uncertainties, we choose zonotopes to represent our samples. A simple case of an uncertain log over a single variable x is depicted in Fig. 1f in the form of simple intervals. The uncertainty can come from the error margin of a sensor: even though the read value is constant, one may need to turn it into an interval, when the sensor only guarantees a limited precision. In addition, the timestamp at each discrete sample of the log can itself be uncertain, in the form of an *interval* (not shown in Fig. 1f, where the timestamp is punctual). This second form of uncertainty can come from network latency, or clocks with limited precision.

Second, to over-approximate the system behavior, and in the spirit of the "model-bounded monitoring" proposed in [WAH22a], we use an extension of *linear dynamical systems*, extended with uncertainty, i.e., allowing *uncertainty* in the dynamics matrix [LP15]. Having some over-approximated knowledge of the system is a natural assumption in practice: when monitoring a car, one generally knows an upper-bound on its maximum speed, or on its maximum acceleration (perhaps depending on its current speed). To cope with the liberal dynamics of our extension of linear dynamical systems, we use a recent technique [GD21b], that performs an efficient reachability analysis for such uncertain linear dynamical systems. The use of such an over-approximation of the actual system is the crux of our approach, allowing us to discard unlikely behaviors, such as the unlikely safety violation depicted in Fig. 1e.

Our first main contribution is to propose a new rigorous analysis technique for offline monitoring of safety properties over scattered *uncertain* samples, using uncertain linear systems as an over-approximation of the system. This over-approximation allows us to extrapolate the behavior since the latest known sample, and to rule out safety violations at some missing discrete samples. Note that our approach uses some discrete analysis as underlying reachability computation technique, and will not however guarantee the absence of safety violations at arbitrary (continuous) timestamps; its main advantage is to offer formal guarantees in the context of missing discrete samples for a given logging granularity.

Our second main contribution focuses on *energy-efficient online monitoring*. For each recorded sample, we run a reachability analysis, and we derive the smallest next discrete time step t in the future at which the safety property may be violated depending on the latest known sample and the over-approximated model dynamics. In a context in which monitoring simply observes the behavior and does not lead to corrective actions, any sample before t is useless because we *know* from the over-approximated model dynamics that no safety violation can happen before t. Therefore, we can schedule the next sample at time t, which reduces the number of discrete samples, and therefore the energy consumption and bandwidth use. We show that our method is correct, i.e., we can safely discard discrete samples without missing any unsafe behavior.

Our third contribution is the implementation of our algorithms into an original tool `MoULDyS` [GA23]. We then show the practical applicability of our approach on three benchmarks: an anesthesia model, an adaptive cruise controller, and an aircraft orbiting system.

We conduct various experiments to showcase the effectiveness and scalability of our approach across multiple factors. Specifically, these experiments highlight how uncertainty in log samples, uncertainty in timestamps, and the number of log samples affect the performance of our algorithms. This encompasses both the scalability of the algorithms and their ability to accurately verify the correctness of the systems behavior.

About this manuscript. This manuscript is an extension of [GA22]. In addition to several details, we significantly increased the content in two main directions. 1) We enhance the uncertainty by considering not only uncertainty over the sample valuations (as in [GA22]), but also over the sample timestamps: in [GA22], the timestamp at each discrete sample of the log was supposed to be constant (i.e., a single point). Here, we extend this notion to an *interval*, making our work able to address a *bi-dimensional uncertainty*. 2) We consider an additional case study of an aircraft orbiting system (new Section 5.4), to which we notably apply our offline algorithm extended with uncertainty over the timestamps. 3) We redid all experiments from [GA22] to remove their randomness: in short, in [GA22], our tool was first generating a random log, and then applying our monitoring algorithms to this log. We decoupled this aspect in the newer version of `MoULDyS` [GA23], and we generated random logs once for all, and then our tool applies monitoring on these statically generated logs—this allows for exact reproducibility of our results.

Outline. We review related works in Section 2. We recall uncertain linear dynamical systems in Section 3. We introduce our offline and online monitoring frameworks in Section 4, and run experiments in Section 5. We draw perspectives in Section 6.

2. Related works

Monitoring. Monitoring complex systems, and notably cyber-physical systems, drew a lot of attention in the last decades, e.g., [MN04, BKZ17, BDD+18, WAH22a, MCW21]. While the main drawback of monitoring is a lack of formal guarantees on the global behavior of a system, its advantage is a much more scalable efficiency compared to techniques such as model checking (see, e.g., [FBCI20]). In addition, monitoring can be performed on black-box systems, the source code (and therefore a model) of which is unavailable. In parallel to monitoring specifications using signal temporal logics (see e.g., [DFM13, JBG+18, QD20]), monitoring using automata-based specifications drew recent attention. Complex, quantitative extensions of automata were studied in the recent years: after timed pattern matching on timed regular expressions [UFAM14] was proposed by Ulus *et al.*, Waga *et al.* proposed a technique for timed pattern matching [WAH16, WHS17, WHS18, Wag19] (with an additional work by Bakhirkin *et al.* [BFN+18]) and then for parametric timed pattern matching [WA19, WAH22b], with application to offline monitoring. Then, techniques for pattern matching were lifted to monitoring against complex specification making use of timing parameters and data parameters [WAH19].

Monitoring cyber-physical systems also shares some similarities (using different techniques and goals) with conformance testing cyber-physical systems (e.g., [Dan11, DMP17, ACM+18]).

In [WAH22a], we proposed *model-bounded monitoring*: instead of monitoring a black-box system against a sole specification, we use in addition a (limited, over-approximated) knowledge of the system, to eliminate false positives. This over-approximated knowledge is

given in [WAH22a] in the form of a *linear hybrid automaton* (LHA) [HPR94], an extension of finite-state automata with continuous variables; their flow in each location ("mode") is given as a linear constraint over derivatives; location invariants and transition guards are given by linear constraints over the system variables. We use in [WAH22a] both an *ad-hoc* implementation, and another one based on PHAVerLite [Fre08, BZ19]. In this work, we share with [WAH22a] the principle of using an over-approximation of the model to rule out some violation of the specification. However, we consider here a different formalism, and we work on discrete samples. In terms of expressiveness of the over-approximated model: *i*) our approach can be seen as less expressive than [WAH22a], in the sense that we have a single (uncertain) dynamics, as opposed to LHAs, where a different dynamics can be defined in each mode; this also allows us to propose a simpler (therefore more efficient) analysis, as each new sample allows us to restart from an exact basis, while in [WAH22a] at each new sample, the system (from an algorithmic point of view) can be in "different modes at the same time"; *ii*) conversely, our dynamics is also significantly more expressive than the LHA dynamics of [WAH22a]; we consider not only the class of linear dynamical systems, but even fit into a special case of non-linear systems, by allowing *uncertainty* in the model dynamics—this is what makes our model an over-approximation of the actual behavior. In addition, we also allow for *uncertain* logs in two dimensions: 1) uncertain *values*—coping with sensor uncertainties, and 2) uncertain *timestamps*—coping with local clock uncertainties and/or network delays. None of these notions of uncertainty were considered in [WAH22a]. We also propose a new *ad-hoc* implementation based on [GD21b].

In [MP16, MP18], a monitor is constructed from a system model in differential dynamic logic [Pla12]. The main difference between [MP16, MP18] and our approach relies in the system model: in [MP16, MP18], the compliance between the model and the behavior is checked at runtime, while our model is assumed to be an over-approximation of the behavior—which is by assumption compliant with the model.

In [SWS21], black-box checking—combining active automata learning and model checking—is improved with specification *strengthening*, increasing the chances to obtain an input violating the specification.

Reachability in linear dynamical systems. In [ALGK11], given a continuous time linear system with input, the system is discretized and reachable sets for consecutive time intervals are computed. At each step, the *state transition matrix* is expressed using the *Peano-Baker* series. The series is then numerically approximated iteratively using *Riemann sums*. Then a zonotope-based convex hull is computed over-approximating the result of all possible matrices in the uncertain matrix. In [CR11], Combastel and Raka extend an existing algorithm based on zonotopes so that it can efficiently propagate structured parametric uncertainties. As a result, they provide an algorithm for computation of envelopes enclosing the possible states and/or outputs of a class of uncertain linear dynamical systems. In [LP15], given an uncertain linear dynamical system $\dot{x} = \Lambda_u x$, Lal *et al.* provide a sampling interval $\delta > 0$, given an $\epsilon > 0$, s.t. the piecewise bilinear function, approximating the solution by interpolating at these sample values, is within ϵ of the original trajectory. [GD19] identifies a class of uncertainties by a set of sufficient conditions on the structure of the dynamics matrix Λ_u. For such classes of uncertainties, the exact reachable set of the linear dynamical system can be computed very efficiently. But this method is not applicable for arbitrary classes of uncertainties. In [GD21b], given an uncertain linear dynamical system, we provide two algorithms to compute reachable sets. The first method is based on perturbation theory,

and the second method leverages a property of linear systems with inputs by representing them as Minkowski sums. In [GD21a], given an uncertain linear dynamical system, we provide an algorithm to compute statistically correct over-approximate reachable sets using *Jeffries Bayes Factor*. Note that uncertain linear dynamical systems are a special subset of non-linear systems. Thus, uncertain linear dynamical systems can also be modeled as a non-linear system. Some additional works that deal with computing reachable sets of non-linear systems are [CÁS13, TD13, CSÁ14, DMVP15, Alt15, KGCC15, CS16].

3. Preliminaries

In this section, we layout the notations and definitions used in the rest of the paper. Formal analysis of safety critical systems requires a precise mathematical model of the system, such as linear dynamical systems. But in reality, the precise, exact model is almost never available—parameter variations, sensor and measurement errors, unaccounted parameters are few such causes that make the availability of a precise model impossible. Presence of such uncertainties in the model makes the safety analysis of these systems useless using traditional methods. Thus, for the analysis to be indeed useful, the safety analysis must consider all possible uncertainties. In [LP15], the authors provide a model, known as *uncertain linear dynamical systems*, to capture such uncertainties. Consider the following example of an uncertain linear dynamical system.

Example 3.1 [GD19, Example 1.1]**.** Let a discrete linear dynamical system $x^+ = \Lambda x$, where $\Lambda = \left[\begin{smallmatrix} 1 & \alpha \\ 0 & 2 \end{smallmatrix}\right]$ and α represents either the modeling uncertainty or a parameter, assuming $2 \leq \alpha \leq 3$. Note that any safety analysis assuming a *fixed* value of α will render the analysis useless—for the safety analysis to be indeed sound, it must consider *all* possible values of α, and they cannot be enumerated.

Intuitively, uncertain linear dynamical systems model the uncertainties in the system by representing all possible dynamics matrices of the system—clearly, this forms a special class of non-linear dynamical systems. To perform safety analysis of uncertain linear dynamical systems, these works provide reachable set computation techniques that account for all possible uncertainties.

Definition 3.2 (Uncertain linear dynamical systems ([GD19, Definition 2.4]))**.** An *uncertain linear dynamical system* is denoted as

$$x^+ = \Lambda x \tag{3.1}$$

where $\Lambda \subset \mathbb{R}^{n \times n}$ is the uncertain dynamics matrix.

Definition 3.3 (Reachable set of an uncertain linear dynamical systems ([GD19, Definitions 2.3 and 2.4]))**.** Given an initial set θ_0 and time step $t \in \mathbb{Z}$, the reachable set of an uncertain linear dynamical system is defined as:

$$RS(\Lambda, \theta_0, t) = \theta_t = \{\theta \mid \theta = \xi_A(\theta_0, t), A \in \Lambda\}. \tag{3.2}$$

where $\xi_A(\theta_0, t) = A^t\theta_0$. An alternative definition is:

$$RS(\Lambda, \theta_0, t) = \theta_t = \bigcup_{A \in \Lambda} \xi_A(\theta_0, t). \tag{3.3}$$

Note that uncertain linear dynamical systems are capable of modeling systems with parameters or when the system dynamics is not perfectly known—the system has modeling uncertainties. [LP15, GD19, GD21b, GD21a] propose various algorithms to compute reachable sets of these systems that account for uncertainties. In this work, we leverage a recently proposed reachable set computation technique, given in [GD21b], to propose our offline and online monitoring algorithm, primarily due to its efficiency vis-à-vis our setting.

Given an initial set $\theta_0 \subset \mathbb{R}^n$ and given a time step t, we denote by $\theta_t \subset \mathbb{R}^n$ the reachable set of the system (given by Eq. (3.1)) at time step t. Next, we define a log of the system with uncertainties in both in system states and timestamps.

Definition 3.4 (Uncertain log)**.** Given an uncertain linear dynamical system as in Eq. (3.1), a finite length *uncertain log* is defined as follows:

$$\ell = \Big\{ \big(\hat{\theta}_t, [t^{lb}, t^{ub}]\big) \mid \theta_t \subseteq \hat{\theta}_t, \text{for some } t \in [t^{lb}, t^{ub}], t^{lb} \leq t^{ub} \leq H \Big\},$$

where H is a given time bound.

Each tuple $\big(\hat{\theta}_t, [t^{lb}, t^{ub}]\big)$ is called a *sample*. Observe that, both the system state $\hat{\theta}_t$ and timestamp $[t^{lb}, t^{ub}]$, in a sample are not necessarily reduced to a *point*. The length of log ℓ—number of samples in ℓ—is given by $|\ell|$. When considering an uncertain log ℓ, the k-th sample from ℓ, where $1 \leq k \leq |\ell|$, can be represented as $\ell_k = \Big(\hat{\theta}_{t_k}, [t_k^{lb}, t_k^{ub}]\Big)$. Here, $\hat{\theta}_{t_k}$ denotes an over-approximation of the system state at a specific time step t_k, where t_k lies within the interval $[t_k^{lb}, t_k^{ub}]$. Such a sample denotes that over-approximate state of the system was observed to be $\hat{\theta}_{t_k}$, for some time step t_k, where $t_k \in [t_k^{lb}, t_k^{ub}]$ (but the exact t_k is not known). This formalism facilitates modeling of situations when the samples are collected over an uncertain channel (such as a shared network with delays) and the precise timestamp is unknown. Note that the log can be *scattered*—it does not necessarily contain a sample for each $t \in \{1, \ldots, H\}$, i.e.,

$$\{1, \cdots, H\} \not\subseteq \bigcup_k [t_k^{lb}, t_k^{ub}].$$

We further note that the uncertainties in the logs, arising from the sensor uncertainties of the logging system, are independent of the uncertainties in the system modeling (Definition 3.2). Given two consecutive samples $\ell_k = \big(\hat{\theta}_{t_k}, [t_k^{lb}, t_k^{ub}]\big)$ and $\ell_{k+1} = \big(\hat{\theta}_{t_{k+1}}, [t_{k+1}^{lb}, t_{k+1}^{ub}]\big)$, we assume that their timestamps do not intersect, i.e., $[t_k^{lb}, t_k^{ub}] \cap [t_{k+1}^{lb}, t_{k+1}^{ub}] = \emptyset$; or, put differently, $t_k^{ub} < t_{k+1}^{lb}$.

When the samples are being collected locally, timestamps can be precisely known (for all practical purposes). In such cases—while we still can have uncertainties in the system state—the timestamp can be known precisely. This leads to a simpler case of logs as defined as follows.

Definition 3.5 (Fixed timestamp uncertain log)**.** A finite length *uncertain log with fixed timestamps* is defined as follows: $\ell = \{(\hat{\theta}_t, t) \mid \theta_t \subseteq \hat{\theta}_t, t \leq H\}$, where H is a given time bound.

We call a fixed timestamp uncertain log ℓ *accurate* if it satisfies the following condition: $\forall 1 \leq k \leq |\ell| : \hat{\theta}_{t_k} = \theta_{t_k}$. Given an uncertain linear dynamical system, $x^+ = \Lambda x$ with an initial set $\theta_0 \subset \mathbb{R}^n$, an *over-approximate reachable set of x^+ at time step t* is $\texttt{overReach}(\Lambda, \theta_0, t)$, such that $\theta_t \subseteq \texttt{overReach}(\Lambda, \theta_0, t)$.

In this work, we use the technique proposed in [GD21b] to compute $\texttt{overReach}(\Lambda, \theta_0, t)$. The algorithm from [GD21b] first computes the reachable set of the nominal dynamics (which excludes uncertainties), and then computes the reachable set related to the uncertainties in the dynamics. These two sets are then combined using the Minkowski sum to obtain the reachable set of the entire dynamics. Although computing the reachable set of the nominal dynamics is straightforward, the reachable set related to uncertainties is challenging to compute. The technique proposed in [GD21b] is sound and demonstrates good scalability, as reported in [GD21b], and confirmed by the experiments conducted in our study (see Section 5). Finally note that any technique can be employed to compute reachable sets of uncertain linear systems as long as it is sound. In other words, as long as the utilized technique is sound, our proposed algorithms remain sound as well.

Safety properties. In this work, we are concerned with *safety* properties. While in many practical cases, a simple threshold over a single variable (or a set of variables) is enough, as in e.g., Fig. 1, we propose a more expressive definition: a safety property is defined as a *zonotope* over the system variables. Since our reachable sets are encoded using zonotopes, safety verification will consist in checking intersection over zonotopes.

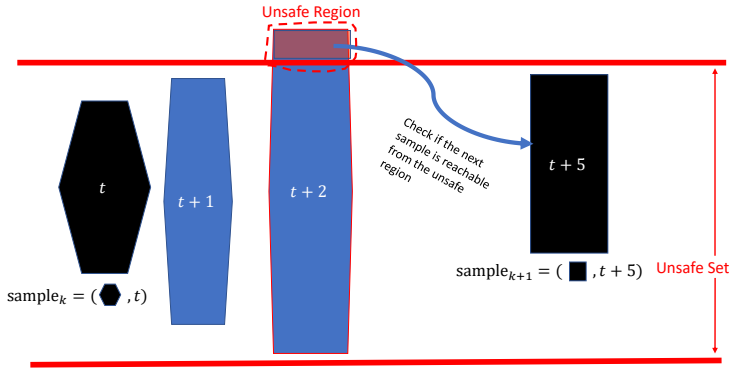
4. Monitoring using uncertain linear dynamical systems as bounding model

In this section, we propose the two main contributions of this work: 1) *Offline monitoring*: Given an uncertain log—arising, e.g., due to faulty sensors and collected over a shared network—we propose an algorithm to infer the safety of a system as given in Eq. (3.1). We prove our method's soundness. 2) *Online monitoring*: We propose a framework to infer safety of a system, as in Eq. (3.1), that triggers the logging system to sample only when needed. Note that, as we only consider the system at *discrete* time steps, the method cannot be sound nor complete, i.e., there always exists a small possibility that the system might violate the safety specification in between two concrete samples (this will be discussed in Section 6.2). However, our online method *is* both *sound* and *complete at the discrete timestamps*, and under the assumption that the samples are free from uncertainties. That is, our method infers the system to be *safe* if and only if the actual behavior of the system is safe at any discrete timestamp, when the logging system can generate accurate samples of the system. Put it differently, we guarantee that skipping some logging in the future using our method will not remove any sample where a violation could have been observed.

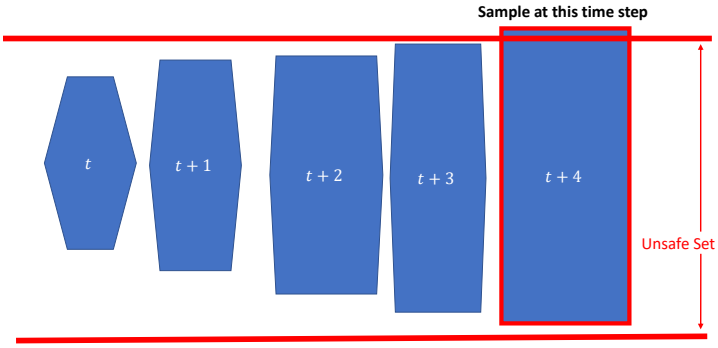
4.1. Offline monitoring over fixed timestamp uncertain logs.

Our first contribution addresses offline monitoring: in this setting, we assume full knowledge of the (possibly scattered) uncertain log, usually after an execution is completely over. In a first step, we assume that logs are known with full certainty regarding the time step; that is, the input is a *fixed timestamp uncertain log*. (The case of a fully uncertain log will be addressed in Section 4.2.)

Before we propose our offline algorithm, we illustrate the approach in Fig. 2a. Consider two consecutive samples k and $k+1$, marked in black, at time steps t and $t+5$ respectively. The reachable sets, in blue, represent the over-approximate behaviors possible by the system between time steps t and $t+5$. Consider the case where at time step $t+2$ the over-approximate reachable set intersects with the unsafe region. Once our algorithm detects a possible unsafe behavior, it computes the intersection between the over-approximate reachable set (here, the



(A) **Offline**



(B) **Online**

FIGURE 2. (2a): **Offline Monitoring**. Black: Two consecutive samples, k and $k+1$, at time steps t and $t+5$ respectively. Blue: The over-approximate reachable set computed from sample k using `overReach`(.). (2b): **Online Monitoring**. Blue: Over-approximate reachable set computed, at each step, using `overReach`(.).

reachable set at time-step $t+2$) and the unsafe set. Then it checks whether the reachable set, given in the next sample $(k+1)$, is reachable from the unsafe region—if yes, it infers *unsafe*; if not, it infers *safe*. Now, we formally propose our offline monitoring method in Algorithm 1 for a given fixed timestamp uncertain log ℓ.

Description. As we first consider an input log given in the form of a fixed timestamp uncertain log (Definition 3.5), we consider a simpler version of Algorithm 1 without the highlighted lines: at line 2, we have $t_k^{lb} = t_k^{ub} = t_k$, and at line 3 we have $t_{k+1}^{lb} = t_{k+1}^{ub} = t_{k+1}$; in addition, the **for** loops at lines 4 and 5 can be discarded.

Let us now describe this simpler version of the algorithm. The **for** loop, starting in line 1, traverses through each sample, and checks if the system can reach a possibly unsafe behavior between two consecutive samples (computed in lines 2 and 3), using over-approximate reachable set computation. If the over-approximate reachable set between two consecutive samples intersect with the unsafe set (line 8), we perform a refinement as follows (line 9–line 13): We compute the unsafe region (intersection between unsafe set and over-approximate reachable set) in line 9, then check if we can reach the next sample from

Algorithm 1: Offline monitoring

input : An uncertain log ℓ of a system $x^+ = \Lambda x$, and an unsafe set $\mathcal{U}$.
output : Return *safe* (resp. *unsafe*) if the actual system behavior is safe (resp. potentially unsafe).

1 **for** $k \in \{1, \ldots, |\ell| - 1\}$ **do**
2 $\quad \left(\hat{\theta}_{t_k}, \left[t_k^{lb}, t_k^{ub}\right]\right) \leftarrow \ell_k$; // current sample, with interval time step
3 $\quad \left(\hat{\theta}_{t_{k+1}}, \left[t_{k+1}^{lb}, t_{k+1}^{ub}\right]\right) \leftarrow \ell_{k+1}$; // next sample, with interval time step
4 $\quad$ **for** $t_k \in \left\{t_k^{lb}, \ldots, t_k^{ub}\right\}$ **do**
5 $\quad\quad$ **for** $t_{k+1} \in \left\{t_{k+1}^{lb}, \ldots, t_{k+1}^{ub}\right\}$ **do**
6 $\quad\quad\quad t_\Delta = t_{k+1} - t_k - 1$; // time gap between two possible time steps of the two samples
$\quad\quad\quad$ /* Compute reachable set for all possible time steps (from the two intervals of time steps) between two samples. Check if any of the sets intersect with the unsafe set */
7 $\quad\quad\quad$ **for** $p \in \{1, \ldots, t_\Delta - 1\}$ **do**
8 $\quad\quad\quad\quad$ **if** $\hat{\theta}_{t_k+p} \cap \mathcal{U} \neq \emptyset$ **then**
$\quad\quad\quad\quad\quad$ /* Refinement process starts */
9 $\quad\quad\quad\quad\quad \psi \leftarrow \hat{\theta}_{t_k+p} \cap \mathcal{U}$; // compute the unsafe region of the system
10 $\quad\quad\quad\quad\quad t_d = t_{k+1} - (t_k + p)$;
11 $\quad\quad\quad\quad\quad \vartheta \leftarrow \texttt{overReach}(\Lambda, \psi, t_d)$;
$\quad\quad\quad\quad\quad$ /* Check if the next sample is reachable from the unsafe region */
12 $\quad\quad\quad\quad\quad$ **if** $\vartheta \cap \hat{\theta}_{t_{k+1}} \neq \emptyset$ **then**
13 $\quad\quad\quad\quad\quad\quad$ **return** *unsafe* ; // the next sample is reachable from the unsafe region
14 $\quad\quad\quad\quad \hat{\theta}_{t_k+p+1} \leftarrow \texttt{overReach}(\Lambda, \hat{\theta}_{t_k+p}, 1)$;
15 **return** *safe* ;

the unsafe region (line 11–line 13). If the next sample is reachable from the unsafe behavior, we conclude the system is unsafe (line 12–line 13).

Soundness and incompleteness. Our proposed offline monitoring approach is *sound* at discrete time steps, but not *complete*—there might be cases where our algorithm returns *unsafe* even though the actual system is *safe*. The primary reason for its incompleteness is due to the fact that `overReach`(.) computes an over-approximate reachable set. Formally:

Theorem 4.1 (soundness at discrete time steps for a fixed timestamp uncertain log)**.** *If the actual system is unsafe at some discrete time step, then Algorithm 1 returns unsafe. Equivalently, if Algorithm 1 returns safe, then the actual system is safe at every discrete time step.*

Proof. We consider a fixed timestamp uncertain log. Let the actual trajectory τ, between two samples k and $k+1$, become unsafe at time step t_{un}. Therefore, the over-approximate reachable set, computed by `overReach`$(\cdot)$ at time step t_{un}, will also intersect with the unsafe

set (due to soundness of `overReach`$(\cdot)$). Note that the actual trajectory τ, originating from the sample k, intersects the unsafe region at time step t_{un}, and reaches the sample $k+1$. The refinement module (Algorithm 1, line 9–line 13), using over-approximate reachable sets will therefore infer the same, concluding the system behavior to be unsafe. □

4.2. **Offline monitoring over uncertain logs.** We now extend our offline monitoring to logs with uncertainty not only in the state dimension, but also in the timestamp dimension (as in Definition 3.4). The extended version of the algorithm is the full Algorithm 1, including the highlighted parts. We namely add a pair of **for** loops at lines 4 and 5, iterating over each (concrete) timestamp in the current uncertain sample ($[t_k^{lb}, t_k^{ub}]$) and over the next uncertain sample ($[t_{k+1}^{lb}, t_{k+1}^{ub}]$). That is, we handle uncertainty over the logging times by iterating over each possible concrete log time in the logged interval. This is a crux to ensure soundness of our approach, and guarantee that a safe answer indeed guarantees safety of the actual system (at all discrete time steps).

Conversely, and as in Section 4.1, our algorithm is not necessarily *complete* (our algorithm might return *unsafe* even though the actual system is *safe*) due to the over-approximation of the reachable set computation.

We prove formally the soundness of Algorithm 1 below:

Theorem 4.2 (soundness at discrete time steps for an uncertain log)**.** *If the actual system is unsafe at some discrete uncertain time step, then Algorithm 1 returns unsafe. Equivalently, if Algorithm 1 returns safe, then the actual system is safe at every discrete uncertain time step.*

Proof. Let the actual trajectory τ, between two samples k and $k+1$, with uncertain time steps $[t_k^{lb}, t_k^{ub}]$ and $[t_{k+1}^{lb}, t_{k+1}^{ub}]$, become unsafe at time step t_{un}. Algorithm 1 computes the reachable set of all possible time steps between $[t_k^{lb}, t_k^{ub}]$ and $[t_{k+1}^{lb}, t_{k+1}^{ub}]$. Therefore, the over-approximate reachable set, computed by `overReach`$(\cdot)$ at time step t_{un}, will also intersect with the unsafe set (due to soundness of `overReach`$(\cdot)$). Note that the actual trajectory τ, originating from the sample k, intersects the unsafe region at time step t_{un}, and reaches the sample $k+1$. The refinement module (Algorithm 1, line 9–line 13), using over-approximate reachable sets will therefore infer the same, concluding the system behavior to be unsafe. □

4.3. **Online monitoring over fixed timestamp uncertain logs.** We now move to *online* monitoring. In contrast to Section 4.2, in our online setting, timestamps are necessarily exact, as we suppose we can trigger (instantaneously) a sample. (Still, there could be cases where uncertainty in the timestamps could be useful in an online setting—this will be discussed in Section 6.2.) However, the logged states are still uncertain (as in Definition 3.5). We propose our online monitoring method in Algorithm 2.

Algorithm 2: Online monitoring

input : An uncertain system $x^+ = \Lambda x$, an unsafe set $\mathcal{U}$, time bound H.
output : Return *safe iff* the actual system behavior is safe.

1 $\hat{\theta}_0 \leftarrow$ Sampling at time step 0 ; // initial behavior of the system.
/* Check whether the initial behavior is safe */
2 **if** $\hat{\theta}_0 \cap \mathcal{U} \neq \emptyset$ **then return** *unsafe* ;
3 **for** $t \in \{1, 2, \ldots, H-1\}$ **do**
4 $\quad \hat{\theta}_{t+1} \leftarrow \texttt{overReach}(\Lambda, \hat{\theta}_t, 1)$; // over-approximate reachable set at next step
$\quad$ /* Check whether the over-approximate reachable set is unsafe */
5 $\quad$ **if** $\hat{\theta}_{t+1} \cap \mathcal{U} \neq \emptyset$ **then**
6 $\quad\quad \ell_{t+1} \leftarrow$ Sample at time step $t+1$;
$\quad\quad$ /* Check whether the actual reachable set is unsafe */
7 $\quad\quad$ **if** $\ell_{t+1} \cap \mathcal{U} \neq \emptyset$ **then**
8 $\quad\quad\quad$ **return** *unsafe* ;
9 $\quad\quad \hat{\theta}_{t+1} = \ell_{t+1}$; // reset to actual behavior
10 **return** *safe*;

Description. The online monitoring algorithm begins by sampling the system at the initial time step, say 0, in line 1. As a sanity check, we confirm if the initial behavior of the system is safe in line 2. The **for** loop starting in line 2—where each iteration corresponds to the set of actions for a time step t—performs the following: At a given time step t, we compute the over-approximate reachable set at the next time step $t+1$ (line 6). If the computed over-approximate reachable set intersects with the unsafe set, we sample the system at time step $t+1$ to check if the actual behavior is also unsafe (line 5–line 9). If safe, we reset the behavior (line 9); if unsafe, we return *unsafe* (line 8). Intuitively, this method samples the actual system only when the over-approximate reachable set, computed by `overReach`(.), intersects the unsafe set. This process is illustrated in Fig. 2b.

Soundness and completeness. Our online monitoring algorithm is correct (safe and complete) at discrete time steps, *provided* the samples are accurate—it returns safe if and only if the actual behavior of the system is safe at all discrete time steps, when accurate samples are obtained. Intuitively, we get the completeness from the fact that it returns unsafe if and only if the (accurate) sample is unsafe. Formally:

Theorem 4.3 (correctness at discrete time steps)**.** *Algorithm 2 returns safe iff the actual behavior at all discrete time steps is safe.*

Proof. The soundness proof—if the actual behavior is unsafe, Algorithm 2 infers unsafe—is straightforward. Hence, we now argue the completeness—if the actual behavior is safe, Algorithm 2 infers *safe*. Note that, Algorithm 2 infers the system behavior as *unsafe* only when a sampled log (actual behavior) becomes unsafe: therefore, if the samples are free from uncertainties (i.e., exact), Algorithm 2 is complete. □

Remark 4.4. While our aim is to consider continuous systems, note that, for *discrete-time systems*, our approach is entirely correct (sound and complete), without the need for a restriction to "discrete time steps", since we can find a granularity small enough for the

discrete-time evolution. This is notably the case for systems where the behavior does not change faster than a given frequency (e.g., the processor clock). In the case of controllers, the granularity can be chosen by selecting the sampling period (the period at which a control input is applied).

Remark 4.5. Given our reliance on enumerating time steps in both offline and online monitoring approaches, an increase in the granularity of sampling periods will necessitate the computation of a larger number of reachable sets. Consequently, this may slow down the analysis process. Nonetheless, the reachable set computation method employed in this work exhibits excellent scalability when dealing with small time steps, as shown in [GD21b]. In other words, if the time interval between two samples in a log is not significantly large (e.g., an order of 500 steps), this technique can easily compute reachable sets. Moreover, if the time gaps do exceed 500 steps, one can enhance the scalability of the reachable set computation by utilizing the interval-based or zonotope-based reduction methods proposed in [GD21b, Section 5.2].

We will study the scalability of our approach in the next section.

5. Case studies

We demonstrate the applicability and usability of our approach on three benchmarks: a medical device (Section 5.2), an adaptive cruise control (Section 5.3), and an aircraft orbiting system (Section 5.4).

5.1. Implementation and environment.

We implemented our offline (both using a fixed timestamp uncertain log and a fully uncertain log) and online monitoring algorithms in a Python-based prototype tool, named `MoULDyS` [GA23]. Tool source and binaries, models and raw results are publicly available on GitHub[2]. Further, the results in this paper can be easily recreated using the scripts provided in the Github repository[3] and the reproducible artifact[4].

Experimental environment. All our experiments were performed on a Lenovo ThinkPad Mobile Workstation with i7-8750H CPU with 2.20 GHz and 32 GiB memory on Ubuntu 20.04 LTS operating system (64 bit). Our tool uses `numpy` [Oli06], `scipy` [VGO+20], `mpmath` [mdt23] for matrix multiplications, [GD21b] to compute `overReach`(.), and the `Gurobi` [GO20] engine for visualization of the reachable sets.

[2] `https://github.com/bineet-coderep/MoULDyS/tree/uncertain_timestamp`

[3] `https://github.com/bineet-coderep/MoULDyS/tree/uncertain_timestamp/src/recreate_results_from_paper`

[4] `https://zenodo.org/doi/10.5281/zenodo.7888501`

Implementation details vis-à-vis Algorithms 1 and 2. The intersection checking between two sets in Algorithms 1 and 2 has been implemented as an optimization formulation in `Gurobi`. That is, given two sets, our implementation of intersection check returns true iff the two sets intersect. In other words, our intersection check is *exact*. In contrast, *computing* the result of the intersection between two sets adds an over-approximation in our implementation. Given two sets, we compute a box hull of the two sets and then compute intersection of the two box hulls. Therefore, the only over-approximate operation we perform in Algorithms 1 and 2—apart from `overReach`$(\cdot)$—is line 9 in Algorithm 1.

Generating scattered uncertain logs for offline monitoring. At each time step, the logging system may take a snapshot of the system evolution at that time step; the logging occurs with a probability p (given). However, it is impossible to determine the precise timestamp of the log if it is being transmitted across a shared network or if the clock tracking the time has errors. Instead, the timestamp then changes into an interval that contains all potential timestamps—this can be referred to as an uncertain timestamp. Given a possible timing delay of t_δ (as per the network's quality), the size of the interval representing the timestamp associated with the log can be anywhere between 0 to t_δ (not necessarily exactly equal to t_δ). In other words, at each such uncertain timestamp, it records the evolution of the system with probability p. Clearly, due to the probabilistic logging, this logger is not guaranteed to generate periodic samples. In each of our three case studies, it is important to highlight that the state variables under monitoring encompass the following: *i*) concentration levels within the anesthesia system, *ii*) distance, acceleration, and velocity within an ACC model, *iii*) positional data of an aircraft. It is evident that in practical scenarios, it becomes nearly impossible to measure the precise values of these variables due to the inherent susceptibility of the sensors used to errors. Nonetheless, considering a specific error margin for a sensor, it becomes feasible to calculate the uncertain value of the variables by adjusting the error tolerance of the sensor. Consequently, we also do not assume that the samples logged by the logging system, at each timestamp, are accurate—the logging system, due to sensor uncertainties, logs an over-approximate sample of the system at that time step. In our experiments, each uncertain log was generated statically from our bounding model (the uncertain linear dynamical system) by simulating its evolution from an uncertain initial set (i.e., not reduced to a point). In the end, we get an uncertain log (as in Definition 3.4).

For the first two out of three case studies, we chose two values for the logging probability p of 20 % and 40 % respectively. They were selected empirically, as our experiments showed that these two values led to quite different behaviors for our offline algorithm: 40 % can be considered as a frequent sampling, while 20 % is more sporadic. We used the same probabilities throughout these two case studies to allow for fair comparison, and for general observations on the effect of sampling probability and uncertainty across experiments.

Logging system for online monitoring. When the logging system is triggered, at a time step, to generate a sample, the logging system records the evolution of the system and sends it to the online monitoring algorithm. Similar to the offline logging system, we do not assume that the samples logged by the logging system are perfectly accurate—the logging system, due to sensor uncertainties, logs an over-approximate sample of the system at that time step. That is, we use the same method—as the offline logging system—to generate logs (statically), but unlike the offline algorithm, the online algorithm uses the samples only when required. For all our case studies, all the generated logs are *safe*.

Research questions. We consider the following research questions in our case studies:

1) Effect of logging probability (number of log samples) on the rate of false alarms raised by the offline monitoring—inferring a behavior as "potentially unsafe" when the actual behavior is "safe".

2) For offline monitoring, does the size of the samples (in other words, volume of the set obtained as sample), gathered at each step, have an impact on the rate of false alarms? Put it differently, what is the effect, *vis-à-vis* false alarms, of the amount of the uncertainty in the log?

3) For online monitoring, how frequent is the logging system triggered to generate a sample?

4) For the same execution, how do the outcome (in terms of verdict on safety by the monitoring algorithms) and the efficiency (in terms of number of samples needed) of the offline and online monitoring algorithms compare?

5) The effect of timing uncertainty on the rate of false alarms raised by the offline monitoring?

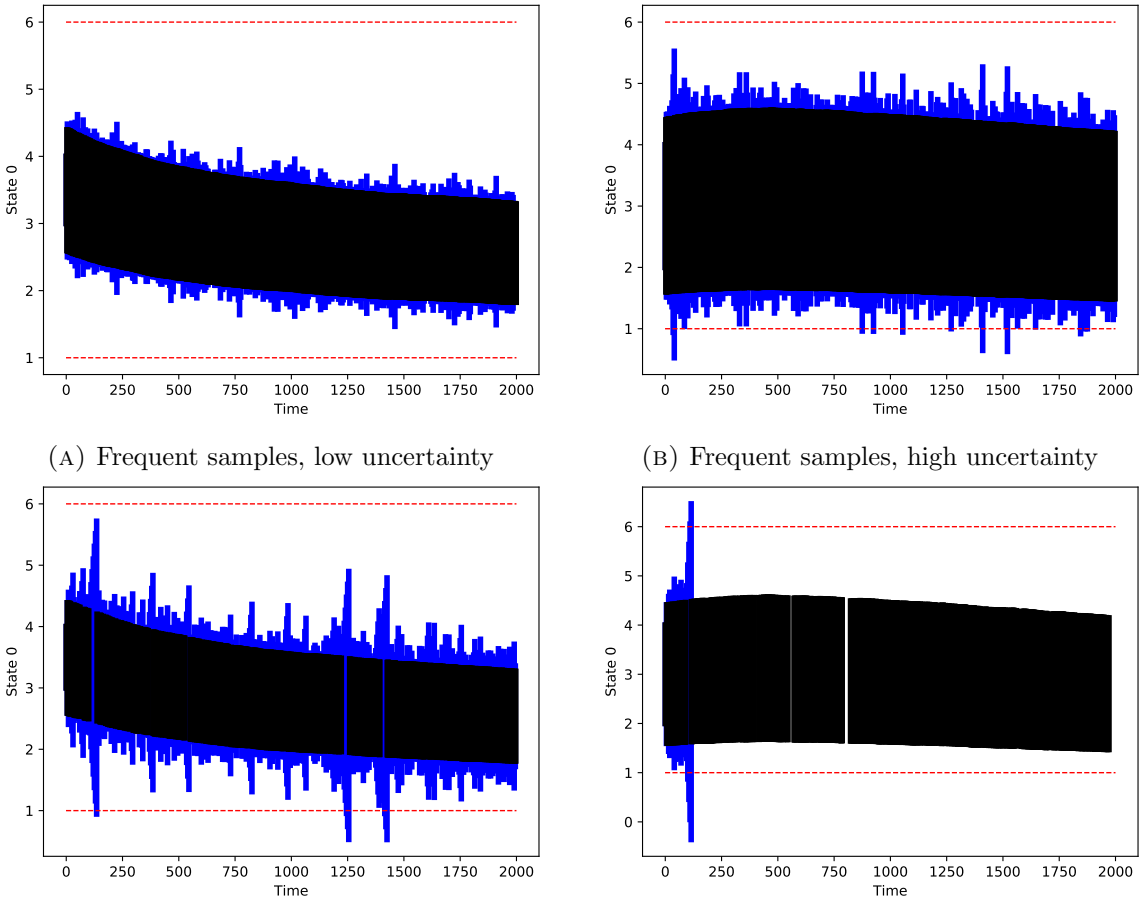
Using the first two case studies, automated anesthetic delivery and adaptive cruise control, Questions (1)-(4) will be addressed. The airplane orbiting benchmark will be used to provide the answer to Question (5). This design decision was made because, unlike the other two benchmarks, only the airplane orbiting benchmark transmits logs through a shared network; in the other two examples, logs are collected locally over a reliable channel.

5.2. **First benchmark: Anesthesia.**

5.2.1. *System description.* We first demonstrate our approach on an automated anesthesia delivery model [GDM14]. The anesthetic drug considered in this model is propofol. Such safety critical systems are extremely important to be verified formally before they are deployed, as under or overdose of the anesthetic drug can be fatal to the patient.

Model. The model as in [GDM14] has two components: 1) Pharmacokinetics (PK): models the change in concentration of the drug as the body metabolizes it. 2) Pharmacodynamics (PD): models the effect of drug on the body. The PK component is further divided into three compartments: *i*) first peripheral compartment c_1, *ii*) second peripheral compartment c_2, *iii*) plasma compartment c_p. The PD component has one compartment, called c_e. The set of state variables of this system is $[\, c_p \; c_1 \; c_2 \; c_e \,]^\top$. The input to the system is the infusion rate of the drug (propofol) u. The complete state-space model of this system in given in [GDM14, Equation 5].

Model parameters. The evolution of state variables c_p, c_1, c_2 is dependent on a number of parameters, such as: the weight of the patient (*weight*), and a number of "first order rate constants" between the compartments (called k_{10}, k_{12}, k_{13}, k_{21} and k_{31} in [GDM14], where their value is given). The evolution of the fourth state variable c_e is dependent on the rate constant between plasma and effect site (called parameter k_d in [GDM14]).



(A) Frequent samples, low uncertainty

(B) Frequent samples, high uncertainty

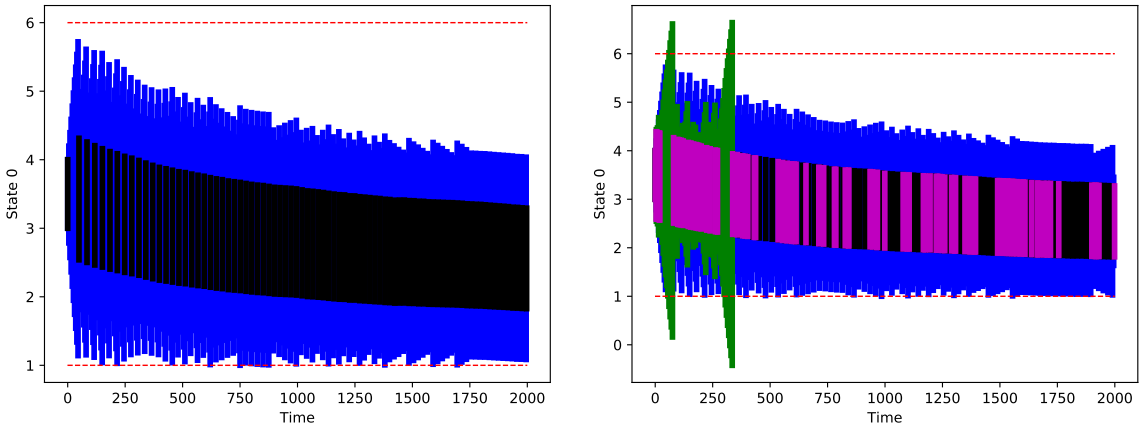
(C) Sporadic samples, low uncertainty

(D) Sporadic samples, high uncertainty

FIGURE 3. ***Offline Monitoring (Anesthesia).*** We plot the change in concentration level of c_p with time. The volume of the samples increases from left to right, and the probability of logging increases from bottom to top. The blue regions are the reachable sets showing the over-approximate reachable sets as computed by the offline monitoring, the black regions are the samples from the log given to the offline monitoring algorithm, and the red dotted line represents safe distance level. Note that although Figs. 3b and 3c reachable sets' seem to intersect with the red line (unsafe set), the refinement module infers them to be *unreachable*, therefore concluding the system behavior as *safe*—unlike Fig. 3d.

Safety. The system is considered safe (as suggested in [GDM14]) if the following concentration levels are maintained at all time steps: $c_p \in [1, 6]$, $c_1 \in [1, 10]$, $c_2 \in [1, 10]$, $c_e \in [1, 8]$. Note that this safety property is a hypercube, i.e., a simple form of a zonotope.

In this case study, we focus our attention on the effect of perturbation, in the weight of the patient (*weight*), on the concentration level of plasma compartment c_p. Only the weight of the patient is subject to perturbation. We assume that the weight of the patient has an additive perturbation of $\pm 0.8\,\text{kg}$ in this case study—at each time step, the weight of the



(A) **Online monitoring** (B) **Comparing online and offline monitoring**

FIGURE 4. ***Online Monitoring (Anesthesia).*** We plot the change in concentration level of c_p with time. The blue regions are the reachable sets showing the over-approximate reachable sets as computed by the online monitoring, the black regions are the samples generated when the logging system was triggered by the online monitoring algorithm, and the red dotted line represents safe concentration levels. *Online monitoring (Fig. 4a):* We apply our online monitoring to the anesthesia model. *Comparison (Fig. 4b):* We compare our online and offline algorithms. The green regions are the reachable sets showing the over-approximate reachable sets between two consecutive samples from the offline logs, the magenta regions are the offline logs, given as an input to the offline monitoring algorithm, generated by the logging system, and the red dotted line represents safe concentration levels. The blue regions are the reachable sets showing the over-approximate reachable sets as computed by the online monitoring, the black regions are the samples generated when the logging system was triggered by the online monitoring algorithm, and the red dotted line represents safe concentration levels.

patient is $weight + \delta_w$, $\delta_w \in [0, 0.8]$. With perturbation in the weight, we want to infer safety of this system using monitoring.

Clearly, monitoring of this system *vis-à-vis* safety is crucial. It is not practical for a busy human doctor or a practitioner to monitor each patient continuously at all time steps—monitoring, either offline or online, provides them an efficient way to save their time and treat their patients without compromising their safety.

The logs for this case study are uncertain "only" in the "valuation" dimension: that is, the logged valuation are known only with a finite precision, but the timestamps are exact; in other words, the input logs are fixed timestamp uncertain logs.

5.2.2. *Experiments.* We now answer questions (1)-(4), using Figs. 3 and 4. In Fig. 3: *i*) the plots in the bottom row have logging probability of 20 %, and the plots in top row have a logging probability of 40 %; *ii*) the plots in left column and the right column have

been simulated with an initial set of $[\,[3{,}4]\ [3{,}4]\ [4{,}5]\ [3{,}4]\,]^\top$, $u \in [2, 5]$ and $[\,[2{,}4]\ [3{,}6]\ [3{,}6]\ [2{,}4]\,]^\top$, $u \in [2, 10]$ respectively. That is, the volume of the samples increases from left to right. In Fig. 4, we simulated the trajectory with an initial set $[\,[3{,}4]\ [3{,}4]\ [4{,}5]\ [3{,}4]\,]^\top$, $u \in [2, 5]$.

Initial state. The initial set is chosen such that the system starts from a safe specification. For each generated log, we start from the initial set, and add the uncertainty depending on our experimental context.

Answer to Question 1. We answer this question by comparing two sets figures in the left column (Figs. 3a and 3c) and the right column (Figs. 3b and 3d) of Fig. 3. *For the left column, i.e., with smaller sample size*: Fig. 3c took 51.40 s and concluded the system to be safe. The analysis in this plot invoked the refinement module of the offline algorithm. But increasing the probability of logging, i.e., more number of samples, as in Fig. 3a, resulted in not invoking the refinement module at all, thus taking 32.92 s. *For the right column, i.e., with larger sample size*: this analysis, as shown in Fig. 3d, took 1.73 s to complete, and concluded the system behavior to be unsafe. The behavior of the system, shown in Fig. 3b with 40 % probability of logging, results in inferring the behavior of the system as safe, by invoking the refinement module several times. Overall, this analysis, as shown in Fig. 3b, took 35.93 s to complete, and concluded the system behavior to be safe.

Answer to Question 2. We answer this question by comparing two sets figures in the top row (Figs. 3a and 3b) and the bottom row (Figs. 3c and 3d) of Fig. 3. *For the bottom row, i.e., with smaller logging probability*: Increasing the volume of the samples results in inferring the behavior from safe (Fig. 3c) to unsafe (Fig. 3d), as per the offline monitoring algorithm. *For the top row, i.e., with higher logging probability*: Increasing the volume of the samples results in not invoking the refinement module (Fig. 3a) to invoking the refinement module several times (Fig. 3b), as per the offline monitoring algorithm.

Answer to Question 3. The result is given in Fig. 4a. Using our online algorithm, we were able to prove safety of the system in 109.04 s. The online algorithm triggered the logging system to generate samples for 83 time steps—this is less than 5 % of total time steps. We observe, as shown in Fig. 4a, that the logging system is triggered more when the trajectory is closer to the unsafe region.

Answer to Question 4. We compare our offline and online algorithms, for 2 000 time steps, on the same trajectory. The result is given in Fig. 4b. Note that, using our online algorithm, we were able to prove safety of the system in 107.99 s. The online algorithm triggered the logging system to generate samples only 84 times. In contrast, the offline algorithm, with a log size of 115 (5 % logging probability) stopped at the 35th sample, (wrongly) inferring the system as unsafe, taking 71.37 s.

5.3. **Second benchmark: Adaptive Cruise Control.** We now apply our offline and online monitoring algorithms to an adaptive cruise control (ACC) model [NHB+16].

5.3.1. *System description.* An adaptive cruise control behaves like an ordinary cruise control when there is no car in the sight of its sensor, and when there is a car in its sight, it maintains a safe distance.

Model. The model as in [NHB+16] has the following state variables: *i*) velocity of the vehicle v, *ii*) distance between the two vehicles h, and *iii*) velocity of the lead vehicle v_L. The state space of the system is given in [NHB+16, Equation 3]. The set of state variables of this system is $[\, v\ h\ v_L \,]^\top$.

Model parameters. The model is dependent on two parameters: *i*) acceleration of the lead vehicle a_L, and *ii*) breaking force and torque applied to the wheels as a lumped net force F. Note that the model is dependent on acceleration of the vehicle a_L, which is very hard to accurately measure due to sensor uncertainties. Similarly the torque F applied to the wheels is also dependent of the coefficient of friction of the ground. To reflect such uncertainties, we consider $a_L \in [-0.9, 0.6]$ and $F \in [-0.6, 2.46]$. These parameters were chosen as per [NHB+16, Tab. 1, Eq. (6)].

Safety. We selected the following safety constraint: The system is considered safe if the distance between vehicles $h > 0.5$. (The unit is, as in [NHB+16], meters.)

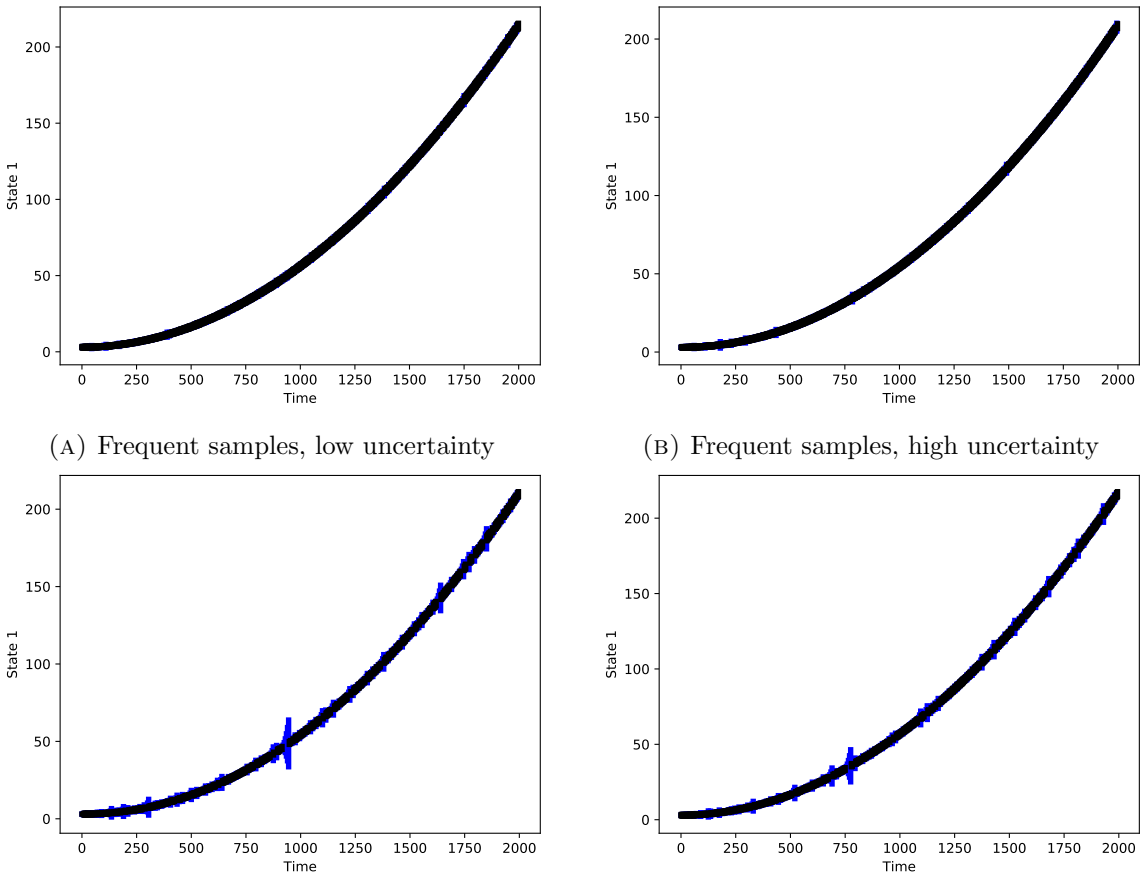
Consider an event of a car crash, where the log stored by the car before the crash, is the only data available to analyze the crash; such an analysis might benefit police, insurance companies, vehicle manufacturers, etc. Using our offline algorithm one can figure out if the car might have shown unsafe behavior or not. Similarly, consider a vehicle on a highway with a lead vehicle in its sight. The ACC in such a case needs to continuously read sensor values to track several parameters, such as acceleration of the lead vehicle, braking force, etc.—this results in wastage of energy. Using our online monitoring algorithm, the car reads sensor values only when there is a potential unsafe behavior. This intermittent behavior will result in saving energy without compromising safety of the system.

With the aforementioned reasons for applying our offline and online monitoring, we apply our algorithms on the ACC model and answer questions (1)-(4). We think that the answers to these questions will help the car designers to design efficient ACC models without compromising safety. Again, the logs for this second case study are uncertain "only" in the "valuation" dimension: that is, they are fixed timestamps uncertain logs.

5.3.2. *Experiments.* Next, we answer questions (1)-(4), using Figs. 5 and 6. In Fig. 5: *i*) the plots in the bottom row have logging probability of 20 %, and the plots in top row have a logging probability of 40 %; *ii*) the plots in left column and the right column have been simulated with an initial set of $[\,[15{,}15.01]\ [3{,}3.03]\ [14.9{,}15]\,]^\top$ and $[\,[15{,}15.1]\ [3{,}3.5]\ [14.9{,}15.1]\,]^\top$ respectively.

Initial state. In Fig. 4, we simulated the trajectory with an initial set $[\,[15{,}15.01]\ [3{,}3.03]\ [14.9{,}15]\,]^\top$, $u \in [2, 5]$. These initial conditions are chosen such that the cars start from a safe specification.

Answer to Question 1. We answer this question by comparing two sets figures in the left column (Figs. 5a and 5c) and the right column (Figs. 5b and 5d) of Fig. 5. *For the left column, i.e., with smaller sample size*: Fig. 5c took 19.08 s and concluded the system to be safe. This analysis in this plot invoked the refinement module of the offline algorithm. But increasing the probability of logging, i.e., more number of samples, as in Fig. 5a, resulted in not invoking the refinement module at all, thus taking 16.5 s. *For the right column, i.e., with larger sample size*: The analysis is similar to that of the left column. Fig. 5d invoked the



(A) Frequent samples, low uncertainty

(B) Frequent samples, high uncertainty

(C) Sporadic samples, low uncertainty

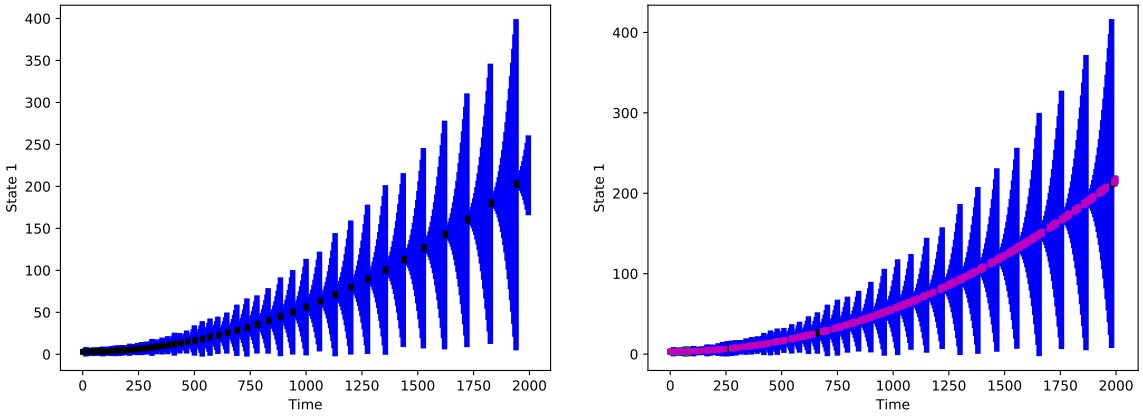
(D) Sporadic samples, high uncertainty

FIGURE 5. ***Offline Monitoring (ACC).*** We plot the change in distance h between the vehicles with time. The volume of the samples increases from left to right, and the probability of logging increases from bottom to top.

refinement module several times, thus taking 20.84 s, while Fig. 5b took 17.5 s, as it invoked the refinement module a smaller number of times.

Answer to Question 2. We answer this question by comparing two sets figures in the top row (Figs. 5a and 5b) and the bottom row (Figs. 5c and 5d) of Fig. 5. *For the bottom row, i.e., with smaller logging probability*: Comparing Fig. 5c and Fig. 5d shows that increasing sample volume results in invoking the refinement module more frequently. A very similar behavior is seen by comparing the top row (i.e., with higher logging probability).

Answer to Question 3. Using our online algorithm, we were able to prove safety of the system in 104.58 s. The online algorithm triggered the logging system to generate samples for 53 time steps—this is less than 3 % of total time steps. This is shown in Fig. 6a.



(A) **Online monitoring** (B) **Comparing online and offline monitoring**

FIGURE 6. ***Online Monitoring (ACC).*** We plot the change in distance between two vehicle h with time. The color coding is same as Fig. 4. *Online monitoring (Fig. 6a)*: We apply our online monitoring to the ACC model. *Comparison (Fig. 6b)*: We compare our online and offline algorithms.

Answer to Question 4. We compare our offline and online algorithm, for 2 000 time steps, on the same trajectory. The result is given in Fig. 6b. Note that, using our online algorithm, we were able to prove safety of the system in 124.46 s. The online algorithm triggered the logging system to generate samples only 50 times. In contrast, the offline algorithm, with a log size of 281 (14 % logging probability) took 28.54 s to infer that the system is safe.

5.4. **Third benchmark: Aircraft Orbiting.** Next, we apply our offline algorithm to an aircraft orbiting case study. Unlike the former two case studies, we consider in this third benchmark not only uncertain valuations, but also uncertain timestamps. That is, logs for this benchmark are uncertain logs (as in Definition 3.4).

5.4.1. *System description.* In this case study, an aircraft is orbiting an object to gather information about the object—depicted in Fig. 7. The aircraft orbits the object by controlling its angular velocity. Since the aircraft operates in an uncertain environment, the angular velocity of the aircraft can have added noise. That is, the model of the aircraft used in this case study can have uncertainties in its angular velocity. The axis of Fig. 7 denotes the position of the aircraft in (x, y) plane (we assume a fixed altitude). The ideal (planned) path of the aircraft, orbiting the object (black), is shown in green. The aircraft must always stay sufficiently close to the object for the data collected about it to be considered reliable. One such bound is shown in the plot with red dashed lines—the aircraft must not cross these lines at any time step. The aircraft transmits its positional data (the x, y values) to the local station at aperiodic intervals. Since the data is transmitted over a shared, long-distance network, a delay is experienced when the data reaches the local station. In other words, the timestamps of the log (behavioral data) can have added noise. Additionally, the behavioral data can also have added noise due to sensor uncertainties. Therefore, this case study is an ideal candidate to demonstrate the applicability of our offline monitoring approach with added noise in timestamps (Section 4.2).

Model. The model as in [LP15, PC09] has the following state variables: *i*) position of the aircraft in two dimensional plane (x_1, x_2), *ii*) and velocity (d_1, d_2). The state space of the system is given in [LP15]. The set of state variables of this system is $[\, x_1 \; x_2 \; d_1 \; d_2 \,]^\top$.

Model parameters. The model is dependent on the angular velocity ω of the aircraft. We assume $\omega \in 1.5 \pm 1\%$.

Safety. The behavior of the aircraft is considered safe, under the presence of model uncertainties, if $x \in [-49.5, 11]$. The safety condition is defined as the state in which the radius of the aircraft orbiting the object remains within a reasonable range. This constraint is necessary to ensure that the sensors can effectively gather data on the object without encountering difficulties caused by an excessively large radius.

5.4.2. *Experiments.* Recall that in the first two case studies (Anesthesia and ACC), we selected logging probabilities of 20 % and 40 % to represent sporadic and frequent logging, respectively. However, in this particular case study, we have chosen a logging probability of 5 % for sporadic logging and 10 % for frequent logging. This choice was made because this case study differs significantly from the two others: through our empirical observations, we found that the former logging probabilities (20 % and 40 %) never provided an opportunity to observe any unsafe behavior in this case study. The computed reachable sets, which represent the possible system states between two logs, were so tightly over-approximated that they never intersected with the unsafe region. Consequently, we had to decrease the logging probability in order to expand the reachable sets and increase the likelihood of identifying potential unsafe conditions. To incorporate this different notion of sporadicity in our experiments, we have empirically chosen logging probabilities of 5 % for sporadic logging and 10 % for frequent logging.

Additionally, to further evaluate the impact of logging probabilities and the uncertainty in sample timestamps, we also conducted experiments using an intermediate logging probability of 7 % with three different choices of timestamp uncertainties. In Fig. 8, the plots in the bottom and top rows have a logging probability of 5 % (sporadic sampling) and 10 % (frequent sampling) respectively, and the plots in the left column and the right column have a timing delay of 2 units and 10 units respectively. Fig. 9 contains unsafe samples and has a logging probability of 10 % (with no time delay), which distinguishes it from the plots in Fig. 8. In Fig. 10, the plots within it have a logging probability of 7 %, while the time delay progressively increases from left to right, with time delays of 2, 6, and 8 units respectively.

Initial state. All the plots in Figs. 8 to 10 have been generated with an initial set of $[\, [1.1,1.11] \; [1.1,1.11] \; [20,20.1] \; [20,20.1] \,]^\top$. The initial set is chosen as per the orbiting path of the aircraft, and such that it remains sufficiently close to the object.

We now answer Questions (1) and (5) using Figs. 8 to 10.

Answer to Question 1. The observations *vis-à-vis* this question are very similar to previous two case studies. We answer this question by comparing two sets figures in the left column (Figs. 8a and 8c) and the right column (Figs. 8b and 8d) of Fig. 8. *For the left column, i.e., with smaller timing delay*: Fig. 8c took 111.6 s and concluded the system to be safe. The analysis in this plot invoked the refinement module of the offline algorithm. But increasing the probability of logging, i.e., more number of samples, as in Fig. 8a, resulted in not invoking

the refinement module at all, thus taking 31.61 s. *For the right column, i.e., with larger timing delay*: this analysis, as shown in Fig. 8d, took 2.33 s to complete, and concluded the system behavior to be unsafe. The behavior of the system, shown in Fig. 8b with 10 % probability of logging, results in inferring the behavior of the system as safe, by invoking the refinement module. Overall, this analysis, as shown in Fig. 8b, took 48.39 s to complete, and concluded the system behavior to be safe. Although none of the plots depicted in Fig. 8 exhibited any unsafe samples, the inclusion of Fig. 9 demonstrates how our offline monitoring approach can trivially identify such unsafe samples and label the system's behavior as unsafe. Fig. 9 accurately labeled the system's behavior as unsafe within a detection time of 4.98 s upon encountering the unsafe sample.

Answer to Question 5. We answer this question by comparing two sets figures in the top row (Figs. 8a and 8b) and the bottom row (Figs. 8c and 8d) of Fig. 8. *For the bottom row, i.e., with smaller logging probability*: Increasing the timing delay of the samples results in inferring the behavior from safe (Fig. 8c) to unsafe (Fig. 8d), as per the offline monitoring algorithm. *For the top row, i.e., with higher logging probability*: Increasing the timing delay the samples results in not invoking the refinement module (Fig. 8a) to invoking the refinement module several times (Fig. 8b), as per the offline monitoring algorithm. The same observation is further reinforced by Fig. 10. Specifically, Figs. 10a to 10c exhibit timing delays of 2, 6, and 8 units respectively, while maintaining the same logging probability across all plots. Notably, we observe that when the timing delays were 2 and 6 units (Figs. 10a and 10b respectively), our offline monitor inferred the behavior to be safe in 48.38 s and 56.48 s respectively. It is worth mentioning that the time required by the offline monitor increases as the time delay of the sample increases. However, in contrast, Fig. 10c illustrates an instance where the offline monitor inferred the system behavior as unsafe, accomplishing this in 4.97 s. Consequently, as also observed in Fig. 8, Fig. 10 demonstrates that an increase in the time delay of the samples can potentially lead to false alarms by the offline monitor, along with an increase in computation time.

Let us now discuss the reasons behind selecting a logging probability of 7 % to generate Fig. 10 (while logging probabilities of 5 % and 10 % were chosen for Fig. 8). A higher logging probability results in fewer false alarms, while a higher timing delay also reduces false alarms. Consequently, with a logging probability of 5 % and a timing delay of 10 units, the system is inferred as unsafe. However, with a logging probability of 10 %, a timing delay of 10 units was successfully verified. In contrast, for Fig. 10, we chose a logging probability of 7 % to empirically determine the tolerable amount of timing delay. Through our experiments, we discovered that with a logging probability of 7 %, the system was able to tolerate timing delays of 2 and 6 units. However, at a timing delay of 8 units, the system was deemed unsafe.

5.5. **General observations.** In the following, we provide general answers to questions (1)-(5) based on our observations from our three case studies.

Answer to Question 1. Increasing the probability of logging reduces the chances of inclusion of spurious behaviors due to over-approximate reachable set computation over longer time horizon. Therefore, it has a reduced chance of spuriously inferring the system unsafe, also fewer chance of invoking the refinement module (as there are less spurious behaviors).

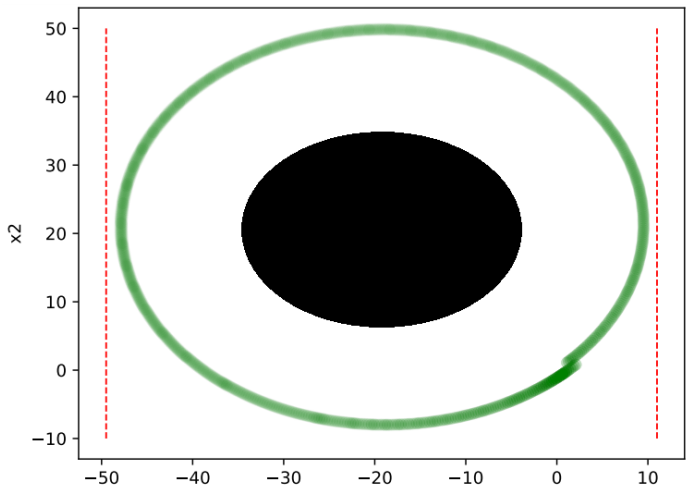


Figure 7. *Planned behavior of the aircraft*: The axis of the plot denotes the position of the aircraft in (x, y) plane (with a fixed altitude). The ideal (planned) path of the aircraft, orbiting the object (black), is shown in green. The red dashed lines indicate the safety constraint of the aircraft—these lines must not be crossed at any time step.

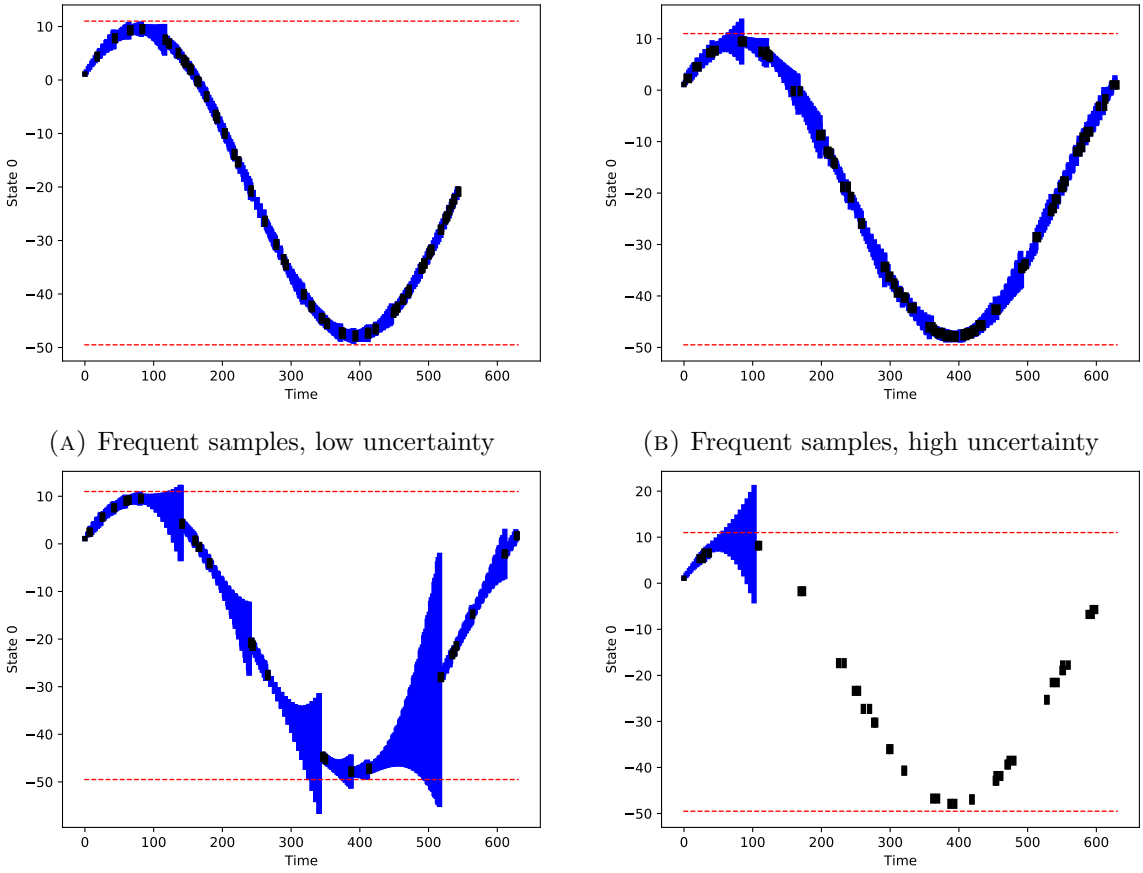
Answer to Question 2. Increasing the size of samples (due to uncertainties or inherent nature of the system) results in increasing chances of invoking the refinement module more frequently. It also increases the chance of (wrongly) inferring the system to be unsafe, as the refinement module can in itself add to the overapproximation.

Answer to Question 3. We observed that our online algorithm is able to prove the system's safety very efficiently with very few samples.

Answer to Question 4. We observed that for a given random log, the offline algorithm was unable to prove safety of the system, whereas our online algorithm was able to prove safety of the system, using fewer samples, by intelligently sampling the system only when needed. We also note that, though here we just demonstrated the result for one random log, but our internal experiments showed that the online algorithm always needed fewer samples to prove safety—which is unsurprising, as it is designed to sample the system only when needed. This can also result in energy saving, as sampling usually requires energy and bandwidth.

Answer to Question 5. Increasing the timing delay of samples results in increasing chances of invoking the refinement module more frequently. It also increases the chance of (wrongly) inferring the system to be unsafe, as the refinement module can in itself add to the overapproximation. Further, it increase in computation time—as it requires exploring all possible combinations of timing delays.

Discussion: Reachable sets computation using `Flow*`. As uncertain linear dynamical systems are a special type of non-linear systems, `Flow*` [CÁS13] would have been a natural candidate to benchmark our offline and online monitoring implementation by comparing various methods to compute `overReach`$(\cdot)$. However, we ran into the following issues: *i*) To



(A) Frequent samples, low uncertainty

(B) Frequent samples, high uncertainty

(C) Sporadic samples, low uncertainty

(D) Sporadic samples, high uncertainty

FIGURE 8. *Offline monitoring (Aircraft Orbiting)*: We plot the position of the aircraft, along x axis, with time. The timing delay of the samples increases from left to right, and the probability of logging increases from bottom to top. The color coding is same as Fig. 3.

the best of our understanding, `Flow*` expects the model of the continuous dynamics to be given as input, along with a discretization parameter. Therefore, trying to encode the time-varying uncertainties in the system as state variables will lead to discretization of the variables encoding uncertainties; such discretization leads to undesired behavior, as those uncertain variables will fail to capture the actual range of values that are possible at any time step. *ii*) However, `Flow*` does allow time varying uncertainties, but only additive[5]. Unfortunately, all our case studies require *multiplicative* uncertainties. Still, we believe `Flow*` could be compared with our implementation when the bounding model has a simpler dynamics than our uncertain linear dynamical systems.

[5]See example at `https://flowstar.org/benchmarks/2-dimensional-ltv-system/`

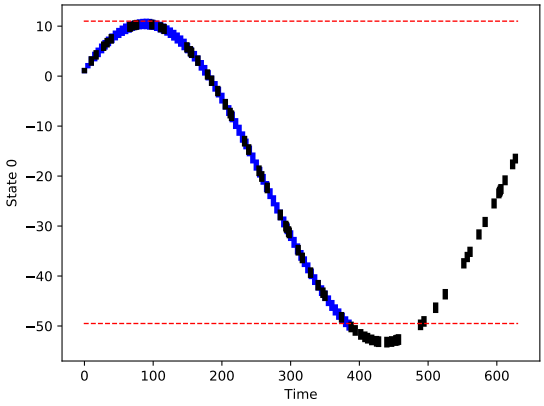


Figure 9. *Offline monitoring with unsafe samples (Aircraft Orbiting):* We plot the position of the aircraft, along x axis, with time. We perform offline monitoring on a log containing unsafe samples. The color coding is same as Fig. 3.

6. Conclusion

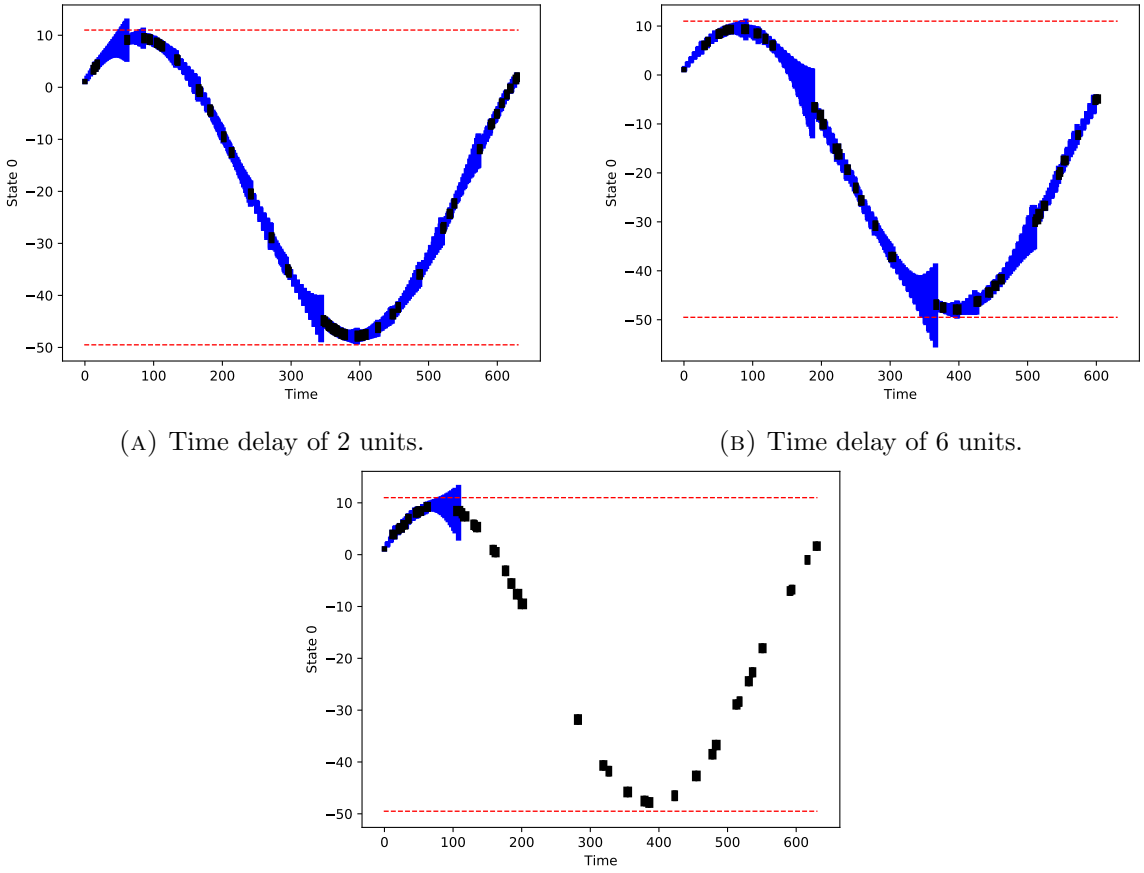
6.1. **Summary.** We presented a new approach for monitoring cyber-physical systems against safety specifications, using the additional knowledge of an over-approximation of the system expressed using an uncertain linear dynamical system. Our approach assumes as first input a log with scattered timestamps (either exact or given in the form of intervals) and uncertain variable samplings (in the form of zonotopes), and as second input an over-approximated model, bounding the possible behaviors. The over-approximation is modeled by *uncertainty* in the variables of the dynamics.

In the offline setting, we are thus able to detect possible violations of safety properties, by extrapolating the known samples with the over-approximated dynamics, and if needed using a second reachability analysis to check whether the next sample is "compatible" with the possible unsafe behavior, i.e., can be reached from the unsafe zone. In the online setting, we are capable of *decreasing* the number of samples, triggering a sample only when there might be a safety violation in a near future, based on the latest known sample and on the over-approximated model dynamics—increasing the energetic efficiency.

Our methods are sound in the sense that an absence of detection of violation by our method indeed guarantees the absence of an actual violation at any discrete time step. In the online method, provided the samples are accurate, our method is in addition complete, i.e., the method outputs *safe* iff the actual system is safe at all discrete time steps. Put it differently, we guarantee that *not* triggering a sample at some time steps is harmless and will not lead to missing a safety violation.

6.2. **Future works.**

Bounding model. The presence of an over-approximated model makes sense, as proposed in [WAH22a]; in our setting of an over-approximated model given by an uncertain linear dynamical system, some formal guarantees that this model indeed represents an over-approximation of the actual system remain to be exhibited.



(A) Time delay of 2 units.

(B) Time delay of 6 units.

(C) Time delay of 8 units.

FIGURE 10. *Effect of timing delay of samples on offline monitoring (Aircraft Orbiting)*: We plot the position of the aircraft, along x axis, with time. The timing delay of the samples increases from left to right, while the probability of logging remains constant at 7 % across all plots. The color coding is same as Fig. 3.

In addition, the assumption of the presence of an over-approximated model is central to our work, and we used it in all our experiments, in the sense that the logs were indeed instances of the over-approximated model. However, an interesting future work will be to partially lift this assumption, by allowing the log to (temporarily, locally) differ from the over-approximated model, allowing for more freedom. In that case, a special care must be made on the approach's soundness.

Enumeration of time steps. A possible threat to validity remains the *enumeration* of time steps in both our algorithms (line 7 in Algorithm 1 and line 3 in Algorithm 2), which could slow down the analysis for very sparse logs—even though this did not seem critical in our experiments. It is worth recalling that the reachable set computation method from [GD21b] that we use in this paper scales very well when computing reachable sets for smaller time steps. Consequently, if the time gap between two samples in a log is not very large (e.g.,

(A) Discrete samples (B) Continuous behavior

FIGURE 11. Incompleteness

an order of 500 steps), this technique computes reachable sets very quickly. Further, if the time gaps become large, one can use the interval-based or the zonotope-based reduction methods proposed in [GD21b, Section 5.2] to improve the scalability of the reachable set computation.

In addition, using skipping methods (as in, e.g., [WHS17]) may help improving the efficiency of our approach.

It would also be interesting to use refinement approaches such as CEGAR [CGJ+00] (Counterexample-Guided Abstraction Refinement) to refine both the time step and the bounding model.

Uncertain timestamps for online monitoring. In contrast to our offline algorithm, our online algorithm monitoring assumes *exact* timestamps: this is not always realistic in all applications. For example, triggering a sample via a shared network, or a long-distance communication (e.g., with a satellite), can take a non-0 time, and result in a sample known with some uncertainty over the timestamp. In that case, a future work is to not wait for the "last second" before triggering a new sample (as in Algorithm 2) but rather trigger a sample Δ time units prior to a possible safety violation, where Δ is some upper bound on the return delay between the monitor and the actual system under monitoring.

Discrete time vs. continuous time. Another future work consists in increasing our guarantees, notably due to the *continuous* nature of cyber-physical systems under monitoring. Indeed, even with a rather fine-grained sampling showing no specification violation (e.g., in Fig. 11a), it can always happen that the actual *continuous* behavior violated the specification (e.g., in Fig. 11b). While setting discrete time steps at a sufficiently fine-grained scale will help to increase the confidence in the results of our approach, no absolutely formal guarantee can be derived. Therefore, one of our future works is to propose some additional conditions for extrapolating (continuous) behaviors between consecutive discrete samples. Also, improving the scope of our guarantees (in the line of, e.g., [DFS21]) is on our agenda.

Finally, in [WAH22a], the bounding model is given using linear hybrid automata, a formalism with a much more restricted dynamics than our approach, but featuring *modes*, i.e., changes of dynamics guarded by some constraints over the variables—which is not considered in our approach. Extending our approach with modes (as in [WAH22a]) is on our agenda, yielding a very expressive bounding model with dynamics beyond linear dynamics, *and* modes. However, this poses some technical difficulties, as the intersection of a set of behaviors with a guard (necessary to check a change of mode) is not proposed by the method from [GD21b]. A future work will be to envision over-approximated intersections.

Acknowledgment

Bineet Ghosh was supported by the National Science Foundation (NSF) of the United States of America under grant number 2038960. Étienne André is partially supported by the ANR-NRF French-Singaporean research program ProMiS (ANR-19-CE25-0015 / 2019 ANR NRF 0092) and ANR BisoUS (ANR-22-CE48-0012).

References

[ACF+21] Étienne André, Emmanuel Coquard, Laurent Fribourg, Jawher Jerray, and David Lesens. Parametric schedulability analysis of a launcher flight control system under reactivity constraints. *Fundamenta Informaticae*, 182(1):31–67, September 2021. doi:10.3233/FI-2021-2065.

[ACM+18] Hugo L. S. Araujo, Gustavo Carvalho, Morteza Mohaqeqi, Mohammad Reza Mousavi, and Augusto Sampaio. Sound conformance testing for cyber-physical systems: Theory and implementation. *Science of Computer Programming*, 162:35–54, 2018. doi:10.1016/j.scico.2017.07.002.

[ALGK11] Matthias Althoff, Colas Le Guernic, and Bruce H. Krogh. Reachable set computation for uncertain time-varying linear systems. In Marco Caccamo, Emilio Frazzoli, and Radu Grosu, editors, *Proceedings of the 14th ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2011)*, pages 93–102. ACM, 2011. doi:10.1145/1967701.1967717.

[Alt15] Matthias Althoff. An introduction to CORA 2015. In Goran Frehse and Matthias Althoff, editors, *Proceedings of the 1st and 2nd International Workshops on Applied veRification for Continuous and Hybrid Systems (ARCH@CPSWeek 2014 and ARCH@CPSWeek 2015)*, volume 34 of *EPiC Series in Computing*, pages 120–151. EasyChair, 2015. doi:10.29007/zbkv.

[BCE+16] David A. Basin, Germano Caronni, Sarah Ereth, Matús Harvan, Felix Klaedtke, and Heiko Mantel. Scalable offline monitoring of temporal specifications. *Formal Methods in System Design*, 49(1-2):75–108, 2016. doi:10.1007/s10703-016-0242-y.

[BCM+92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992. doi:10.1016/0890-5401(92)90017-A.

[BDD+18] Ezio Bartocci, Jyotirmoy V. Deshmukh, Alexandre Donzé, Georgios E. Fainekos, Oded Maler, Dejan Nickovic, and Sriram Sankaranarayanan. Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications. In Ezio Bartocci and Yliès Falcone, editors, *Lectures on Runtime Verification – Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*, pages 135–175. Springer, 2018. doi:10.1007/978-3-319-75632-5_5.

[BFN+18] Alexey Bakhirkin, Thomas Ferrère, Dejan Nickovic, Oded Maler, and Eugene Asarin. Online timed pattern matching using automata. In David N. Jansen and Prabhakar Pavithra, editors, *Proceedings of the 16th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2018)*, volume 11022 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2018. doi:10.1007/978-3-030-00151-3_13.

[BKZ17] David A. Basin, Felix Klaedtke, and Eugen Zalinescu. The MonPoly monitoring tool. In Giles Reger and Klaus Havelund, editors, *Proceedings of An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CuBES 2017)*, volume 3 of *Kalpa Publications in Computing*, pages 19–28. EasyChair, 2017.

[BZ19] Anna Becchi and Enea Zaffanella. Revisiting polyhedral analysis for hybrid systems. In Bor-Yuh Evan Chang, editor, *Proceedings of the 26th International Symposium on Static Analysis (SAS 2019)*, volume 11822 of *Lecture Notes in Computer Science*, pages 183–202. Springer, 2019. doi:10.1007/978-3-030-32304-2_10.

[CÁS13] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In Natasha Sharygina and Helmut Veith, editors, *Proceedings of the 25th International Conference on Computer Aided Verification (CAV 2013)*, volume 8044 of *Lecture Notes in Computer Science*, pages 258–263. Springer, 2013. doi:10.1007/978-3-642-39799-8_18.

[CGJ+00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and A. Prasad Sistla, editors, *Proceedings of*

the 12th International Conference on Computer Aided Verification (CAV 2000), volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000. doi:10.1007/10722167_15.

[CKNZ11] Edmund M. Clarke, William Klieber, Milos Novácek, and Paolo Zuliani. Model checking and the state explosion problem. In Bertrand Meyer and Martin Nordio, editors, *Revised Tutorial Lectures from the LASER International Summer School 2011 (Tools for Practical Software Verification)*, volume 7682 of *Lecture Notes in Computer Science*, pages 1–30. Springer, 2011. doi:10.1007/978-3-642-35746-6_1.

[CR11] Christophe Combastel and Sid-Ahmed Raka. On computing envelopes for discrete-time linear systems with affine parametric uncertainties and bounded inputs. *IFAC Proceedings Volumes*, 44(1):4525–4533, 2011. Proceedings of the 18th IFAC World Congress. doi:10.3182/20110828-6-IT-1002.02585.

[CS16] Xin Chen and Sriram Sankaranarayanan. Decomposed reachability analysis for nonlinear systems. In *Proceedings of the 2016 IEEE Real-Time Systems Symposium (RTSS 2016)*, pages 13–24. IEEE Computer Society, 2016. doi:10.1109/RTSS.2016.011.

[CSÁ14] Xin Chen, Sriram Sankaranarayanan, and Erika Ábrahám. Under-approximate flowpipes for non-linear continuous systems. In *Proceedings of the Formal Methods in Computer-Aided Design (FMCAD 2014)*, pages 59–66. IEEE, 2014. doi:10.1109/FMCAD.2014.6987596.

[Dan11] Thao Dang. Model-based testing of hybrid systems. In Justyna Zander, Ina Schieferdecker, and Pieter J. Mosterman, editors, *Model-Based Testing for Embedded Systems*, Computational Analysis, Synthesis, & Design Dynamic Systems. CRC Press, 2011. doi:10.1201/b11321-15.

[DFM13] Alexandre Donzé, Thomas Ferrère, and Oded Maler. Efficient robust monitoring for STL. In Natasha Sharygina and Helmut Veith, editors, *Proceedings of the 25th International Conference on Computer Aided Verification (CAV 2013)*, volume 8044 of *Lecture Notes in Computer Science*, pages 264–279. Springer, 2013. doi:10.1007/978-3-642-39799-8_19.

[DFS21] Johann C. Dauer, Bernd Finkbeiner, and Sebastian Schirmer. Monitoring with verified guarantees. In Lu Feng and Dana Fisman, editors, *Proceedings of the 21st International Conference on Runtime Verification (RV 2021)*, volume 12974 of *Lecture Notes in Computer Science*, pages 62–80. Springer, 2021. doi:10.1007/978-3-030-88494-9_4.

[DMP17] Jyotirmoy V. Deshmukh, Rupak Majumdar, and Vinayak S. Prabhu. Quantifying conformance using the Skorokhod metric. *Formal Methods in System Design*, 50(2-3):168–206, 2017. doi:10.1007/s10703-016-0261-8.

[DMVP15] Parasara Sridhar Duggirala, Sayan Mitra, Mahesh Viswanathan, and Matthew Potok. C2E2: A verification tool for stateflow models. In Christel Baier and Cesare Tinelli, editors, *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2015), Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS 2015)*, volume 9035 of *Lecture Notes in Computer Science*, pages 68–82. Springer, 2015. doi:10.1007/978-3-662-46681-0_5.

[FBCI20] Mohammed Foughali, Saddek Bensalem, Jacques Combaz, and Félix Ingrand. Runtime verification of timed properties in autonomous robots. In *Proceedings of the 18th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE 2020)*, pages 1–12. IEEE, 2020. doi:10.1109/MEMOCODE51338.2020.9315156.

[Fre08] Goran Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. *International Journal on Software Tools for Technology Transfer*, 10(3):263–279, May 2008. doi:10.1007/s10009-007-0062-x.

[GA22] Bineet Ghosh and Étienne André. Monitoring of scattered uncertain logs using uncertain linear dynamical systems. In Mohammad Mousavi and Anna Philippou, editors, *Proceedings of the 42nd International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE 2022)*, volume 13273 of *Lecture Notes in Computer Science*, pages 67–87. Springer, 2022. doi:10.1007/978-3-031-08679-3_5.

[GA23] Bineet Ghosh and Étienne André. MoULDyS: Monitoring of autonomous systems in the presence of uncertainties. *Science of Computer Programming*, 230, August 2023. doi:10.1016/j.scico.2023.102976.

[GD19] Bineet Ghosh and Parasara Sridhar Duggirala. Robust reachable set: Accounting for uncertainties in linear dynamical systems. *ACM Transactions on Embedded Computing Systems*, 18(5s):97:1–97:22, 2019. doi:10.1145/3358229.

[GD21a] Bineet Ghosh and Parasara Sridhar Duggirala. Reachability of linear uncertain systems: Sampling based approaches. Technical Report 2109.07638, arXiv, 2021. URL: https://arxiv.org/abs/2109.07638, arXiv:2109.07638.

[GD21b] Bineet Ghosh and Parasara Sridhar Duggirala. Robustness of safety for linear dynamical systems: Symbolic and numerical approaches. Technical Report 2109.07632, arXiv, 2021. URL: https://arxiv.org/abs/2109.07632, arXiv:2109.07632.

[GDM14] Victor Gan, Guy Albert Dumont, and Ian Mitchell. Benchmark problem: A PK/PD model and safety constraints for anesthesia delivery. In Goran Frehse and Matthias Althoff, editors, *Proceedings of the 1st and 2nd International Workshops on Applied veRification for Continuous and Hybrid Systems (ARCH@CPSWeek 2014 and ARCH@CPSWeek 2015)*, volume 34 of *EPiC Series in Computing*, pages 1–8. EasyChair, 2014. doi:10.29007/8drm.

[GO20] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*, 2020. URL: http://www.gurobi.com.

[HPR94] Nicolas Halbwachs, Yann-Éric Proy, and Pascal Raymond. Verification of linear hybrid systems by means of convex approximations. In Baudouin Le Charlier, editor, *Proceedings of the First International Static Analysis Symposium (SAS 1994)*, volume 864 of *Lecture Notes in Computer Science*, pages 223–237. Springer, 1994. doi:10.1007/3-540-58485-4_43.

[JBG+18] Stefan Jakšić, Ezio Bartocci, Radu Grosu, Thang Nguyen, and Dejan Ničković. Quantitative monitoring of STL with edit distance. *Formal Methods in System Design*, 53(1):83–112, 2018. doi:10.1007/s10703-018-0319-x.

[KGCC15] Soonho Kong, Sicun Gao, Wei Chen, and Edmund M. Clarke. dReach: δ-reachability analysis for hybrid systems. In Christel Baier and Cesare Tinelli, editors, *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2015), Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS 2015)*, volume 9035 of *Lecture Notes in Computer Science*, pages 200–205. Springer, 2015. doi:10.1007/978-3-662-46681-0_15.

[KGN+09] Roope Kaivola, Rajnish Ghughal, Naren Narasimhan, Amber Telfer, Jesse Whittemore, Sudhindra Pandav, Anna Slobodová, Christopher Taylor, Vladimir A. Frolov, Erik Reeber, and Armaghan Naik. Replacing testing with formal verification in Intel CoreTM i7 processor execution engine validation. In Ahmed Bouajjani and Oded Maler, editors, *Proceedings of the 21st International Conference on Computer Aided Verification (CAV 2009)*, volume 5643 of *Lecture Notes in Computer Science*, pages 414–429. Springer, 2009. doi:10.1007/978-3-642-02658-4_32.

[LLN18] Kim Guldstrand Larsen, Florian Lorber, and Brian Nielsen. 20 years of UPPAAL enabled industrial model-based validation and beyond. In Tiziana Margaria and Bernhard Steffen, editors, *Proceedings of the 8th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice (ISoLA 2018), Part IV*, volume 11247 of *Lecture Notes in Computer Science*, pages 212–229. Springer, 2018. doi:10.1007/978-3-030-03427-6_18.

[LP15] Ratan Lal and Pavithra Prabhakar. Bounded error flowpipe computation of parameterized linear systems. In Alain Girault and Nan Guan, editors, *Proceedings of the 2015 International Conference on Embedded Software (EMSOFT 2015)*, pages 237–246. IEEE, 2015. doi:10.1109/EMSOFT.2015.7318279.

[Mal16] Oded Maler. Some thoughts on runtime verification. In Yliès Falcone and César Sánchez, editors, *Proceedings of the 16th International Conference on Runtime Verification (RV 2016)*, volume 10012 of *Lecture Notes in Computer Science*, pages 3–14. Springer, 2016. doi:10.1007/978-3-319-46982-9_1.

[MCW21] Konstantinos Mamouras, Agnishom Chattopadhyay, and Zhifu Wang. A compositional framework for quantitative online monitoring over continuous-time signals. In Lu Feng and Dana Fisman, editors, *Proceedings of the 21st International Conference on Runtime Verification (RV 2021)*, volume 12974 of *Lecture Notes in Computer Science*, pages 142–163. Springer, 2021. doi:10.1007/978-3-030-88494-9_8.

[mdt23] The mpmath development team. *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 1.3.0)*, 2023. https://mpmath.org/.

[MN04] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In Yassine Lakhnech and Sergio Yovine, editors, *Proceedings of the Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS 2004) and Formal Techniques*

in Real-Time and Fault-Tolerant Systems (FTRTFT 2004), volume 3253 of *Lecture Notes in Computer Science*, pages 152–166. Springer, 2004. doi:10.1007/978-3-540-30206-3_12.

[MP16] Stefan Mitsch and André Platzer. ModelPlex: verified runtime validation of verified cyber-physical system models. *Formal Methods in System Design*, 49(1-2):33–74, 2016. doi:10.1007/s10703-016-0241-z.

[MP18] Stefan Mitsch and André Platzer. Verified runtime validation for partially observable hybrid systems. Technical report, 2018. URL: http://arxiv.org/abs/1811.06502, arXiv:1811.06502.

[NHB+16] Petter Nilsson, Omar Hussien, Ayca Balkan, Yuxiao Chen, Aaron D. Ames, Jessy W. Grizzle, Necmiye Ozay, Huei Peng, and Paulo Tabuada. Correct-by-construction adaptive cruise control: Two approaches. *IEEE Transactions on Control Systems Technology*, 24(4):1294–1307, 2016. doi:10.1109/TCST.2015.2501351.

[Oli06] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[PC09] André Platzer and Edmund M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In Ana Cavalcanti and Dennis Dams, editors, *Proceedings of the Second World Congress on Formal Methods (FM 2009)*, volume 5850 of *Lecture Notes in Computer Science*, pages 547–562. Springer, 2009. doi:10.1007/978-3-642-05089-3_35.

[Pel08] Radek Pelánek. Fighting state space explosion: Review and evaluation. In Darren D.and Alessandro Fantechi Cofer, editor, *Proceedings of the 13th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2008)*, volume 5596 of *Lecture Notes in Computer Science*, pages 37–52. Springer, 2008. doi:10.1007/978-3-642-03240-0_7.

[Pla12] André Platzer. The complete proof theory of hybrid systems. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LICS 2012)*, pages 541–550. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.64.

[QD20] Xin Qin and Jyotirmoy V. Deshmukh. Clairvoyant monitoring for signal temporal logic. In Nathalie Bertrand and Nils Jansen, editors, *Proceedings of the 18th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2020)*, volume 12288 of *Lecture Notes in Computer Science*, pages 178–195. Springer, 2020. doi:10.1007/978-3-030-57628-8_11.

[SWS21] Junya Shijubo, Masaki Waga, and Kohei Suenaga. Efficient black-box checking via model checking with strengthened specifications. In Lu Feng and Dana Fisman, editors, *Proceedings of the 21st International Conference on Runtime Verification (RV 2021)*, volume 12974 of *Lecture Notes in Computer Science*, pages 100–120. Springer, 2021. doi:10.1007/978-3-030-88494-9_6.

[TD13] Romain Testylier and Thao Dang. NLTOOLBOX: A library for reachability computation of nonlinear dynamical systems. In Dang Van Hung and Mizuhito Ogawa, editors, *Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA 2013)*, volume 8172 of *Lecture Notes in Computer Science*, pages 469–473. Springer, 2013. doi:10.1007/978-3-319-02444-8_37.

[UFAM14] Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. Timed pattern matching. In Axel Legay and Marius Bozga, editors, *Proceedings of the 12th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2014)*, volume 8711 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2014. doi:10.1007/978-3-319-10512-3_16.

[VGO+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi:10.1038/s41592-019-0686-2.

[WA19] Masaki Waga and Étienne André. Online parametric timed pattern matching with automata-based skipping. In Julia Badger and Kristin Yvonne Rozier, editors, *Proceedings of the 11th Annual NASA Formal Methods Symposium (NFM 2019)*, volume 11460 of *Lecture Notes in Computer Science*, pages 371–389. Springer, 2019. doi:10.1007/978-3-030-20652-9_26.

[Wag19] Masaki Waga. Online quantitative timed pattern matching with semiring-valued weighted automata. In Étienne André and Mariëlle Stoelinga, editors, *Proceedings of the 17th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS*

2019), volume 11750 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2019. doi:10.1007/978-3-030-29662-9_1.

[WAH16] Masaki Waga, Takumi Akazaki, and Ichiro Hasuo. A Boyer-Moore type algorithm for timed pattern matching. In Martin Fränzle and Nicolas Markey, editors, *Proceedings of the 14th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2016)*, volume 9884 of *Lecture Notes in Computer Science*, pages 121–139. Springer, 2016. doi:10.1007/978-3-319-44878-7_8.

[WAH19] Masaki Waga, Étienne André, and Ichiro Hasuo. Symbolic monitoring against specifications parametric in time and data. In Işil Dillig and Serdar Tasiran, editors, *Proceedings of the 31st International Conference on Computer-Aided Verification (CAV 2019), Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 520–539. Springer, 2019. doi:10.1007/978-3-030-25540-4_30.

[WAH22a] Masaki Waga, Étienne André, and Ichiro Hasuo. Model-bounded monitoring of hybrid systems. *ACM Transactions on Cyber-Physical Systems*, 6(4):30:1–30:26, November 2022. doi:10.1145/3529095.

[WAH22b] Masaki Waga, Étienne André, and Ichiro Hasuo. Parametric timed pattern matching. *ACM Transactions on Software Engineering and Methodology*, 32(1):10:1–10:35, February 2022. doi:10.1145/3517194.

[WHS17] Masaki Waga, Ichiro Hasuo, and Kohei Suenaga. Efficient online timed pattern matching by automata-based skipping. In Alessandro Abate and Gilles Geeraerts, editors, *Proceedings of the 15th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2017)*, volume 10419 of *Lecture Notes in Computer Science*, pages 224–243. Springer, 2017. doi:10.1007/978-3-319-65765-3_13.

[WHS18] Masaki Waga, Ichiro Hasuo, and Kohei Suenaga. MONAA: A tool for timed pattern matching with automata-based acceleration. In *Proceedings of the 3rd Workshop on Monitoring and Testing of Cyber-Physical Systems (MT@CPSWeek 2018)*, pages 14–15. IEEE, 2018. doi:10.1109/MT-CPS.2018.00014.