`MoULDyS`: Monitoring of Autonomous Systems in the Presence of Uncertainties

Bineet Ghosh[a], Étienne André[b,c]

[a] *The University of North Carolina at Chapel Hill, NC, USA*
[b] *Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France*
[c] *Université Sorbonne Paris Nord, LIPN, CNRS UMR 7030, F-93430 Villetaneuse, France*

Abstract

We introduce `MoULDyS`, that implements efficient offline and online monitoring algorithms of black-box cyber-physical systems w.r.t. safety properties. `MoULDyS` takes as input an uncertain log (with noisy and missing samples), as well as a bounding model in the form of an uncertain linear system; this latter model plays the role of an over-approximation so as to reduce the number of false alarms. `MoULDyS` is Python-based and available under the *GNU General Public License v3.0* (`gpl-3.0`). We further provide easy-to-use scripts to recreate the results of two case studies introduced in an earlier work.

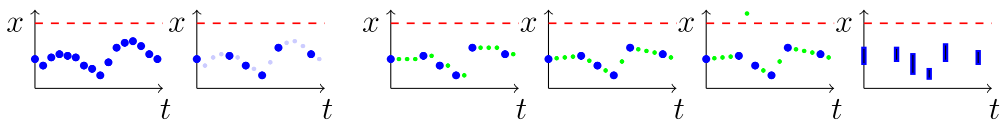
Keywords: energy-aware monitoring, cyber-physical systems, formal methods, monitoring tool.

1. Motivation and Significance

Monitoring consists of analyzing system logs, e.g., for detecting safety violations (see, e.g., [4]). Monitoring has many useful applications such as detecting the cause of a crash of vehicles. As an example, autonomous systems are generally equipped with a device that records their state at periodic or aperiodic time steps—logging the behavior of the system until the time of a failure. A log, comprising of such recorded samples, is then investigated for possible safety violations. Not only the logs can have samples missing at various time steps, but also the recorded samples can have added noise to it, e.g., due to sensor uncertainties. Analyzing such logs to detect possible safety violations, that might have caused a failure, is known as *offline monitoring* when the analysis is done *a posteriori* (see, e.g., [5]). In contrast, it is *online* when performed on-the-fly, when the whole log is not (yet) known (see [6] for a discussion on online verification).

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v1.1
C2	Permanent link to code/repository used for this code version	`https://github.com/bineet-coderep/MoULDyS/releases/tag/v1.1`
C3	Permanent link to Reproducible Capsule	`10.5281/zenodo.7888502`
C4	Legal Code License	*GNU General Public License v3.0* (`gpl-3.0`) [1]
C5	Code versioning system used	`git`
C6	Software code languages, tools, and services used	Python, `numpy`, `scipy`, `mpmath`, `pandas`, `Gurobi`
C7	Compilation requirements, operating environments and dependencies	Provided in the installation guide [2].
C8	If available, link to developer documentation/manual	Provided in the user guide [3].
C9	Support email for questions	`bineet@cs.unc.edu`

Table 1: Code metadata



(a) Full samples (b) Monitored (c) Extrapol (d) Extrapol (e) Safety (f) Uncertain

Figure 1: Monitoring at discrete time steps [9]

We introduce here `MoULDyS`[1] (see Table 1 for code metadata), a monitoring tool to analyze logs to detect possible safety violations. The specific features of `MoULDyS` are twofold: 1) the possibility to monitor aperiodic logs, or periodic logs with missing samples, and with possible noise over the recorded data; and 2) the presence of a bounding model following the formalism of uncertain linear systems.

Uncertain linear systems. Uncertain linear systems [7, 8] are a special subclass of non-linear systems that can be used to represent uncertainties and parameters in linear dynamical systems—for example, they can represent uncertain parameters in the cells of the dynamics matrix. Such a formalism is useful in representing an over-approximation of the model when the precise model is unknown or hard to obtain.

[1] `https://sites.google.com/view/mouldys`

Monitoring using uncertain linear systems as a bounding model. A challenge when performing offline monitoring is to "recreate" or guess the samples at the missing time steps. That is, when the system under monitoring is a black-box, with a log in the form of an aperiodic timed sequence of valuations of continuous variables (with missing valuations at various time steps): *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?* Consider Fig. 1a, a system for which a logging occurs at every discrete time step. When logging occurs at only *some* time steps (due to some sensor faults, or to save energy with only a sparse, scattered logging), a possible output is in Fig. 1b. In such a setting, *how to be certain that, in between two discrete samples, another discrete sample (not recorded) did not violate the specification?* That is, 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—it can yield a "safe" answer, while the actual system could have actually 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. Without any knowledge of the model, one can always assume that the behavior given in Fig. 1e could happen. This behavior shows that the variable x is suddenly 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". To alleviate such issues, we proposed in [9] an offline monitoring algorithm using a *bounding model*, i.e., a rough overapproximation of the system behavior (originally introduced in [10] in a different context). The proposed method is based on the reachable set computation of uncertain linear systems [11] that can detect safety violations with limited false alarms.

We also considered in [9] an *online* monitoring algorithm, aiming at energetic efficiency, by recording samples only when required (i.e., when the system may get closer to a violation). `MoULDyS` implements both our offline and online monitoring algorithms [9]. We also provide here the steps to easily recreate the results of the two case studies in [9].

Experimental setting. Given an aperiodic (i.e., missing valuations at various time steps) and a noisy log (i.e., the valuations that are present in the log can have an added noise—an overapproximation of the actual state), `MoULDyS` can perform offline monitoring of the system to detect safety of the system behavior. Further, MoULDyS can be used in an online setting to log only when necessary—thus targeting energy efficiency while logging. MoULDyS can be run on a standard laptop with a Linux operating system (see the installation guide for details [2]). The details of how to use the tool, with illustrative examples, can be found in the user guide [3].

Outline. Section 2 describes the software; Section 3 describes the architecture; Section 4 describes the functionalities; Section 5 describes two illustrative examples from medical and automotive domains, with necessary steps to recreate their results; Section 6 discusses the impact of MoULDyS; Section 7 briefly reviews related works; Section 8 concludes and discusses future research.

2. Software Description

MoULDyS is an open-source software, implemented in Python, running on Linux platforms. Our experiments [9] suggest that MoULDyS is able to perform monitoring of reasonably large systems in a reasonable time. For example, we are able to monitor a five-dimensional system (i.e., a system with 5 continuous variables monitored at the same time) for 2000 time steps, with only actual 300 samples (note that, the fewer samples, the higher is the monitor computation time—as it is required to "recreate" the missing samples using reachable sets) in under 2.5 minutes on a standard laptop.

We believe that MoULDyS will not just be helpful to engineers to analyze logs to detect safety violations in several areas of research (such as in robotics to detect collision and other undesirable behaviors), but also to researchers to further develop monitoring-based approaches—in that case MoULDyS can be used for comparison.

3. Software Architecture

The architecture of MoULDyS is given in Fig. 2. Each block in Fig. 2 represents a core part (either functional or input) of the tool, and the arrows indicate the flow of data. The dataflow of MoULDyS, for both Figs. 3 and 4 (offline and online monitoring respectively), starts from the blocks in the left (e.g., bounding model, unsafe set, etc.) and ends at the extreme right of the figures (outputting the safety status, visualization, and/or synthesized log). Since both the offline (`Block 1` of Figs. 2 and 3) and online monitoring algorithm (`Block 2` in Figs. 2 and 4) uses reachability of uncertain linear systems (`Block 3` of Fig. 2), a data exchange occurs between `Block 1` and `Block 3`, as well as `Block 2` and `Block 3`. MoULDyS implements a built-in reachability

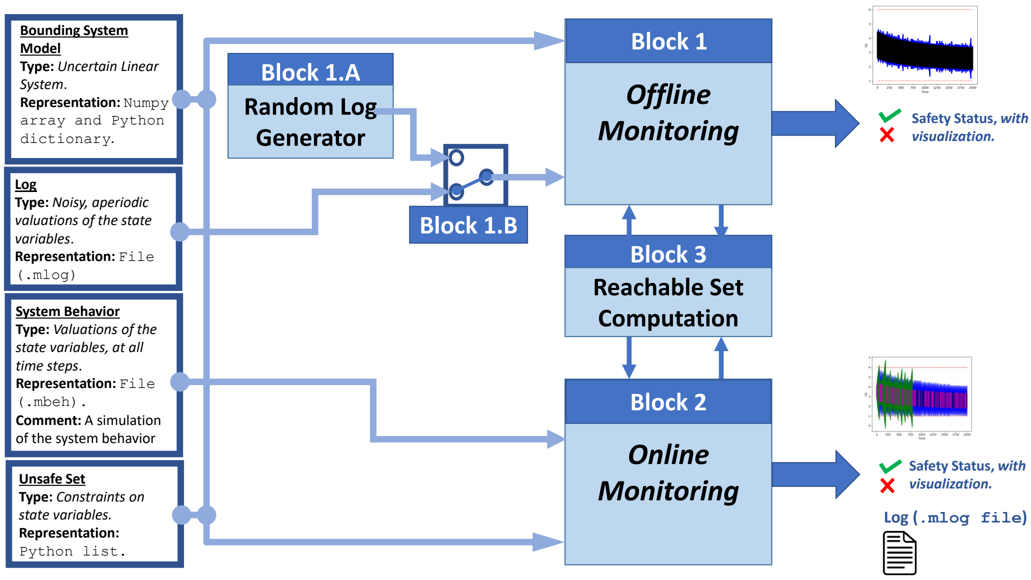


Figure 2: `MoULDyS` ***Architecture.*** Each block identifies a core part of the tool, and the arrows indicate the flow of data.

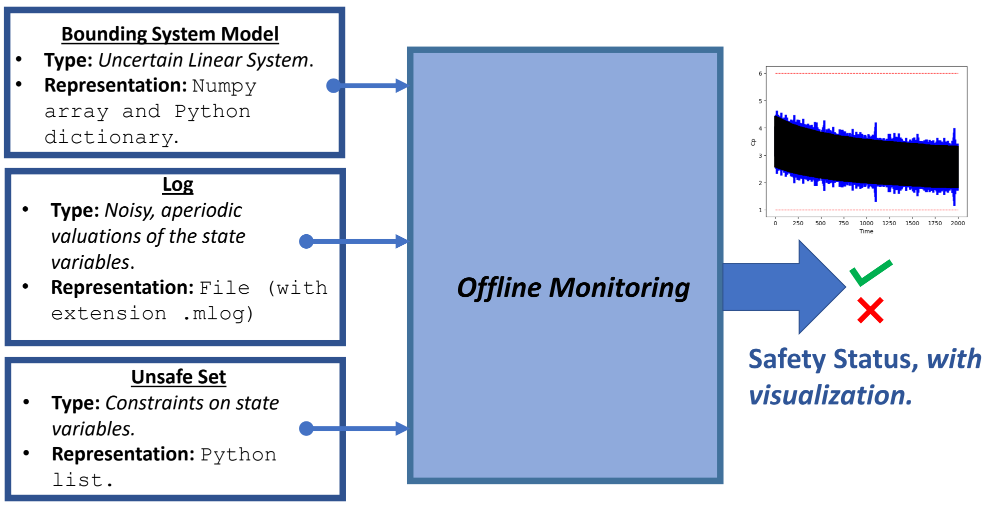


Figure 3: Dataflow diagram of the offline monitoring functionality of `MoULDyS`.

algorithm. The native reachable set computation support facilitates faster computing (i.e., no data exchange with third party tool is required, which would potentially require additional data reformatting) with no additional tool installation required. `MoULDyS` employs an algorithm proposed in [11] to

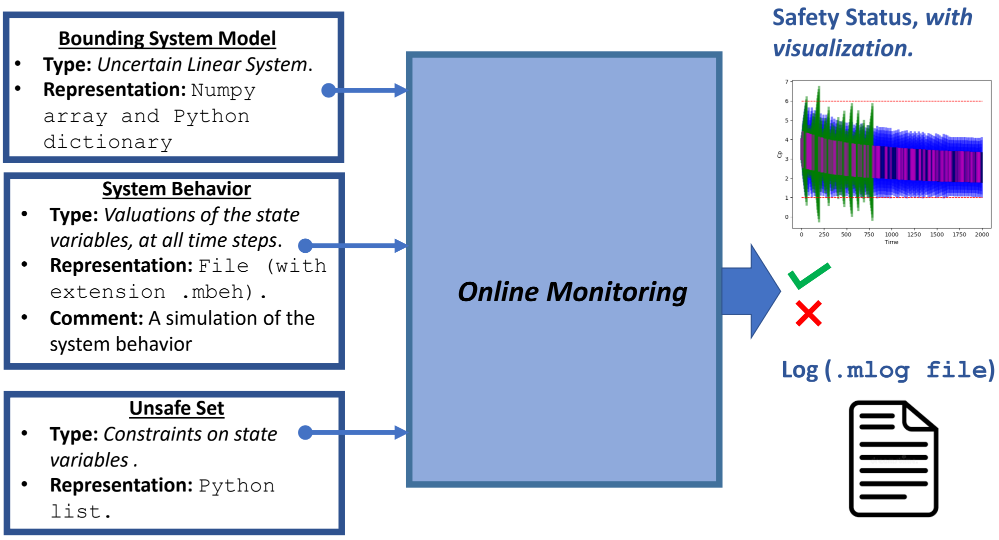


Figure 4: Dataflow diagram of the online monitoring functionality of `MoULDyS`.

compute the reachable set of uncertain linear systems. The algorithm 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. After obtaining the reachable sets, `MoULDyS` verifies the safety of these sets by comparing them against provided safety specifications. These safety specifications are constraints on state variables, which can be complex and involve multiple state variables (e.g., linear inequalities involving several state variables). To represent such safety specifications, `MoULDyS` uses a special type of polytope called *zonotopes*, which can be expressed as an affine transformation of a unit box. `MoULDyS` can check multiple safety specifications, each represented as a zonotope and involving several state variables.

3.1. Implementation

`MoULDyS` is implemented using `Python 3.7.x`, and runs in a Linux environment. The architecture is given in Fig. 2.

The tool can be used in the following two ways. On the one hand, users can use it through the provided virtual machine[2], which already contains all the necessary dependencies and has the path variable set. Nevertheless, users are still required to obtain and install the Gurobi license themselves, since Gurobi only grants free academic licenses to individuals. On the other hand, the tool can also be downloaded and setup from its public GitHub repository[3]. If users aim to recreate the results in the paper or simply employ it for basic monitoring purposes, using the provided virtual machine is recommended. However, if the tool will be used for research and development purposes, it is recommended to download and set it up on a local machine. The detailed installation instructions are provided in [2]. The system model (represented as an uncertain linear system), in `MoULDyS`, is represented with a `numpy` array and a dictionary. The unsafe set is represented with a Python list. An example code of how to encode the system model and its unsafe specification can be found in the public repository[4]. The log and the system behavior, on the other hand, is given as a file to `MoULDyS` (`MoULDyS` can also generate random logs; discussed later). Logs can either be represented as zonotopes or intervals (an example of the required file can be found in `MoULDyS` public repository[5]).

Both the online and offline monitoring algorithm (`Block 1` and `Block 2` of Fig. 2) have been implemented in Python using standard libraries, namely `numpy`, `scipy` and `mpmath`. The reachable set computation (`Block 3` of Fig. 2) module implements the algorithm proposed in [11] in Python. Both the online and the offline module require performing intersection checking of zonotopes [9], which has been implemented as an optimization formulation using `Gurobi`. `Gurobi` has been further used to visualize the reachable sets.

Installation. While the virtual machine comes preinstalled with the required dependencies, setting up `MoULDyS` on a local machine requires installing the following dependencies: `numpy`, `scipy`, `mpmath`, `pandas`, `Gurobi` along with `gurobipy` (`Gurobi` requires an *ad-hoc* install but is free to use for academic purposes). The detailed steps for installing `MoULDyS` are given in the installation guide [2].

User Guide. A tutorial on how to use several functionalities of `MoULDyS`, along with sample codes (encoding a toy dynamics), is given in the user

[2] 10.5281/zenodo.7888502

[3] https://github.com/bineet-coderep/MoULDyS/releases/tag/v1.1

[4] https://www.github.com/bineet-coderep/MoULDyS/blob/main/src/tutorial/TutorialOfflineMonitoring.py

[5] https://www.github.com/bineet-coderep/MoULDyS/blob/main/data/toyEg_5_interval.mlog

guide [3].

4. Software Functionalities

In this section, we discuss the core functionalities of `MoULDyS`:

Offline Monitoring The offline monitoring requires the bounding model of the system (represented as an uncertain linear system) to be given as input. Further, a log of the system behavior is required, which can be achieved by the following ways: *i*) If the user already has a log to be monitored, it can be simply passed as an input to `Block 1` of Fig. 2. *ii*) Alternatively, `MoULDyS` can also generate a random (noisy and aperiodic) log of the system, from a given initial set, using `Block 1.A`. The selection between the two possible choices is facilitated by `Block 1.B`. Analyzing the log, the final output of the offline monitoring is either `safe` (indicating the system behavior is certainly safe at all time steps), or `possibly-unsafe` (indicating the system might have shown unsafe behavior). An example code snippet to perform offline monitoring, on a toy example, can be found in its public repository.[6]

Online Monitoring The online monitoring requires the bounding model of the system (represented as an uncertain linear system) to be given as input, as well as the actual behavior of the system. The actual behavior of the system is given as a file (with extension `.mbeh`) representing the values of the state variables at every time step—an example of the expected file (with extension `.mbeh`), containing valuation of the state variables at every time step, can be found in its public repository.[7] The final output of this feature is the safety status of the system (`safe`/`possibly-unsafe`), and a synthesized log. An example code snippet to perform online monitoring, on a toy example, can be found in its public repository.[8]

As a side functionality, `MoULDyS` also allows to generate random logs (using `Block 1.A`). While this is not strictly speaking part of monitoring, it greatly helps to perform experiments using `MoULDyS`. Basically, given a

[6] https://www.github.com/bineet-coderep/MoULDyS/blob/main/src/tutorial/TutorialOfflineMonitoring.py

[7] https://www.github.com/bineet-coderep/MoULDyS/blob/main/data/toyEg_5_interval.mbeh

[8] https://www.github.com/bineet-coderep/MoULDyS/blob/main/src/tutorial/TutorialOnlineMonitoring.py

bounding model, `MoULDyS` can generate a random log following the bounding model.

5. Illustrative examples

In this section, we briefly recall the two case studies presented in [9] that use a prototype version of `MoULDyS`. The two case studies, automated anesthesia delivery and adaptive cruise control, demonstrate the applicability and usability of `MoULDyS`. Further, we provide detailed steps to recreate the results presented in [9] using `MoULDyS`.

Anesthesia [12] presents an automated anaesthesia delivery model, with the drug propofol. The system models the metabolization of the drug by the body, and the depth of hypnosis. The state variables encode the various concentration levels—that must be within a certain limit at all times—modeling the metabolization of the drug and the depth of hypnosis. Note that a higher concentration level would mean that the patient remains unconscious for a longer period of time, while a lower concentration level would mean that patient remains conscious during the surgery—which can be traumatic. `MoULDyS` can help performing an automated monitoring of the patients without compromising on safety.

Adaptive Cruise Control (ACC) [13] presents a model of ACC with state variables as velocity, distance between two vehicles, and the velocity of the lead vehicle. Offline monitoring provides an automated way to detect the *cause of the crash* and *who was at fault*. Similarly, consider a vehicle driving on a highway with a vehicle in its sight. The ACC unit will have to continuously read sensor values to track several parameters, such as acceleration of the lead vehicle, braking force, etc.—causing a waste of energy. In these cases, deploying online monitoring on the vehicle ACC will ensure that the sensor values are only read when there is a potential unsafe behavior—thus saving energy. [9] provides several such practical cases where monitoring is useful—in [9, Figs. 5 and 6] (also recalled in the appendix in Figs. B.8 and B.9 respectively).

The case studies mentioned above study the following aspects with regards to monitoring: *i*) Impact of number of samples in the log. *ii*) Impact of uncertainties in the samples of the log. *iii*) Demonstrating online monitoring. *iv*) Comparing offline and online monitoring. In the following, we provide the detailed steps to recreate the results in [9, Section 5]. In particular, the results that we wish to recreate, from [9], are given in Figs. B.6 to B.9.

5.1. Recreating Results

The results of the Anesthesia case study and the ACC case study can be recreated by using the scripts provided in the GitHub repository[9]. The detailed steps to recreate the results from both the case studies are given in [14].

Note that, in [9, Section 5], logs were randomly generated *during our experiments*, and therefore the results from [9, Section 5] cannot be *stricto sensu* be recreated, as their reproducibility script re-generates a random log, which may differ from the one actually shown in [9, Section 5].

Therefore, we modified our scripts so that the log is given as an input, on which monitoring is performed: we thus generated the logs statically, and embedded them in the reproducibility capsule, in order for users to reproduce exactly the result we present here. These logs were generated with the same logging probabilities (and initial sets) as in [9, Section 5]. In the rest of the section, we discuss the steps for replicating the new Figs. B.6 to B.9.

Anesthesia. The main results of the Anesthesia case study are provided in Figs. B.6 and B.7 (variant of [9, Figures 3 and 4]). We use `python Anesthesia.py -offline i`, with `i` $\in \{1, 2, 3, 4\}$ to recreate Figs. B.6a to B.6d respectively. To recreate Fig. B.7a, we use `python Anesthesia.py -online`. To recreate Fig. B.7b, we use `python Anesthesia.py -compare`. The `Anesthesia.py` script is provided in its GitHub repository.[10]

ACC. The main results of the ACC case study are provided in Figs. B.8 and B.9 (variant of [9, Figures 5 and 6]). The results of the ACC case study can be recreated in a similar manner to the Anesthesia case study, by simply using the `ACC.py`[11] script instead of the `Anesthesia.py` script.

6. Impact

While `MoULDyS` remains a prototype based on a recent algorithm [9], and is by no means a widely used software in an industrial context for the time being, we believe it has an interesting potential to gain up a user base interested in monitoring black-box cyber-physical systems against safety properties. To the best of our knowledge, it is the first software allowing monitoring logs

[9] https://www.github.com/bineet-coderep/MoULDyS/tree/main/src/recreate_results_from_paper

[10] https://www.github.com/bineet-coderep/MoULDyS/blob/main/src/recreate_results_from_paper/Anesthesia.py

[11] https://github.com/bineet-coderep/MoULDyS/blob/main/src/recreate_results_from_paper/ACC.py

featuring not only uncertainty (due to sensor's imperfect behavior) but also potentially missing samples, together with a bounding model going beyond the class of linear systems. This is in contrast with, e.g., [10] in which our bounding model was restricted to linear models.

In addition, `MoULDyS` can perform *online* monitoring, with a focus on energetic efficiency: by triggering a sample recording only when necessary (i.e., when `MoULDyS` informs the system that it may get close to an unsafe behavior according to its online algorithm), the system saves energy, i.e., only records samples (which needs network bandwidth usage, as well as processor and memory usage) when necessary instead of at every time unit.

Our applications recalled in Section 5 show that `MoULDyS` can be applied to challenging domains such as health and autonomous driving, giving interesting results (e.g., limited number of false alarms) in a reasonable execution time making it suitable to real-time applications.

7. Related works

Monitoring complex systems, and notably cyber-physical systems, drew a lot of attention in the last decades [4]. We briefly review close works in the following.

MonPoly [15] is a monitoring tool taking as specification formulas expressed using MFOTL (metric first-order temporal logic). It is entirely black-box: the only input beyond the formula is the *log*, i.e., a sequence of timestamped system events, potentially with numeric arguments (e.g., "`@10 withdraw (Alice,6000)`", expressing that a withdrawal occurs at timestamp 10).

In [16], the focus is on online monitoring over real-valued signals, using MTL as the specification formalism. Again, the system is black-box.

In [17], parametric timed pattern matching is made, on an entirely black-box system, i.e., without any prior knowledge of the system; the tools used are IMITATOR [18] and a prototypal tool ParamMONAA. The output is a set of intervals where a property is valid/violated, possibly with a set of timing parameter valuations.

In [10], 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 [10] in the form of a *linear hybrid automaton* (LHA) [19]. We use in [10] both an *ad-hoc* implementation, and another one based on PHAVerLite [20]. In this work, we share with [10] the principle of using an over-approximation of the model to rule out some violation of the specification, which comes in contrast with the aforementioned works. However, we consider here a different formalism, and we work on discrete samples. In terms of expressiveness of the over-approximated model, while our approach can be seen as less expressive than [10], in the sense that we have a single (uncertain) dynamics (as opposed to LHAs, where a different dynamics can be defined in each mode), our dynamics is also significantly *more expressive* than the LHA dynamics of [10]; 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.

In [21, 22], a monitor is constructed from a system model in differential dynamic logic [23]. The main difference between [21, 22] and our approach relies in the system model: in [21, 22], 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.

8. Conclusion and Future Work

Monitoring black-box complex cyber-physical systems can be delicate, and may lead to false alarms. `MoULDyS` is a Python-based tool implementing offline and online monitoring algorithms. A first crux of `MoULDyS` is to be able to manage logs with uncertainty over the logged state variables, as well as missing samples. A second crux is the use of a bounding model in the form of uncertain linear systems, helping to reduce the number of false alarms. `MoULDyS` can analyze logs efficiently to detect possible safety violations that might have caused an unsafe behavior. Further, `MoULDyS` can also be used in an online setting where the system is sampled only when there is a risk of safety violation. As a result, the online monitoring is able to decrease the number of samples, therefore reducing energy consumption at runtime. `MoULDyS` is available under the GNU General Public License.

In future, we wish to extend `MoULDyS` to support uncertainty not only in the log valuations (the value of a sensor at a given timestamp), but also uncertainty in the log timestamps themselves: this makes sense when some sensors are distributed with drifting clocks, or when network delays make the exact recording timestamp imprecise.

Acknowledgements (optional)

Bineet Ghosh was supported by the National Science Foundation (NSF) of the United States of America under grant number 2038960. This work is partially supported by the ANR-NRF French-Singaporean research program

ProMiS (ANR-19-CE25-0015 / 2019 ANR NRF 0092) and by ANR BisoUS (ANR-22-CE48-0012).

References

[1] GNU General Public License v3.0, https://www.gnu.org/licenses/gpl-3.0.en.html.

[2] MoULDyS installation guide, https://www.github.com/bineet-coderep/MoULDyS/blob/main/documentation/installation_guide.md (2022).

[3] MoULDyS user guide, https://www.github.com/bineet-coderep/MoULDyS/blob/main/documentation/user_guide.pdf (2022).

[4] E. Bartocci, J. V. Deshmukh, A. Donzé, G. E. Fainekos, O. Maler, D. Ničković, S. Sankaranarayanan, Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications, in: E. Bartocci, Y. Falcone (Eds.), Lectures on Runtime Verification – Introductory and Advanced Topics, Vol. 10457 of Lecture Notes in Computer Science, Springer, 2018, pp. 135–175. doi:10.1007/978-3-319-75632-5_5.

[5] D. A. Basin, G. Caronni, S. Ereth, M. Harvan, F. Klaedtke, H. Mantel, Scalable offline monitoring of temporal specifications, Formal Methods in System Design 49 (1-2) (2016) 75–108. doi:10.1007/s10703-016-0242-y.

[6] O. Maler, Some thoughts on runtime verification, in: Y. Falcone, C. Sánchez (Eds.), RV, Vol. 10012 of Lecture Notes in Computer Science, Springer, 2016, pp. 3–14. doi:10.1007/978-3-319-46982-9_1.

[7] R. Lal, P. Prabhakar, Bounded error flowpipe computation of parameterized linear systems, in: A. Girault, N. Guan (Eds.), EMSOFT, IEEE, 2015, pp. 237–246. doi:10.1109/EMSOFT.2015.7318279.

[8] B. Ghosh, P. S. Duggirala, Robust reachable set: Accounting for uncertainties in linear dynamical systems, ACM Transactions on Embedded Computing Systems 18 (5s) (2019) 97:1–97:22. doi:10.1145/3358229.

[9] B. Ghosh, É. André, Monitoring of scattered uncertain logs using uncertain linear dynamical systems, in: M. Mousavi, A. Philippou (Eds.), FORTE, Vol. 13273 of Lecture Notes in Computer Science, Springer, 2022, pp. 67–87. doi:10.1007/978-3-031-08679-3_5.

[10] M. Waga, É. André, I. Hasuo, Model-bounded monitoring of hybrid systems, ACM Transactions on Cyber-Physical Systems 6 (4) (2022) 30:1–30:26. doi:10.1145/3529095.

[11] B. Ghosh, P. S. Duggirala, Robustness of safety for linear dynamical systems: Symbolic and numerical approaches, Tech. Rep. 2109.07632, arXiv (2021).

[12] V. Gan, G. A. Dumont, I. Mitchell, Benchmark problem: A PK/PD model and safety constraints for anesthesia delivery, in: G. Frehse, M. Althoff (Eds.), ARCH@CPSWeek, Vol. 34 of EPiC Series in Computing, EasyChair, 2014, pp. 1–8. doi:10.29007/8drm.

[13] P. Nilsson, O. Hussien, A. Balkan, Y. Chen, A. D. Ames, J. W. Grizzle, N. Ozay, H. Peng, P. Tabuada, Correct-by-construction adaptive cruise control: Two approaches, IEEE Transactions on Control Systems Technology 24 (4) (2016) 1294–1307. doi:10.1109/TCST.2015.2501351.

[14] MoULDyS: Recreating results, https://www.github.com/bineet-coderep/MoULDyS/blob/main/documentation/recreate_results.md (2022).

[15] D. A. Basin, F. Klaedtke, E. Zalinescu, The MonPoly monitoring tool, in: G. Reger, K. Havelund (Eds.), RV-CuBES, Vol. 3 of Kalpa Publications in Computing, EasyChair, 2017, pp. 19–28.

[16] K. Mamouras, A. Chattopadhyay, Z. Wang, A compositional framework for quantitative online monitoring over continuous-time signals, in: L. Feng, D. Fisman (Eds.), RV, Vol. 12974 of Lecture Notes in Computer Science, Springer, 2021, pp. 142–163. doi:10.1007/978-3-030-88494-9_8.

[17] M. Waga, É. André, I. Hasuo, Parametric timed pattern matching, ACM Transactions on Software Engineering and Methodology 32 (1) (2022) 10:1–10:35. doi:10.1145/3517194.

[18] É. André, IMITATOR 3: Synthesis of timing parameters beyond decidability, in: R. Leino, A. Silva (Eds.), CAV, Vol. 12759 of Lecture Notes in Computer Science, Springer, 2021, pp. 1–14. doi:10.1007/978-3-030-81685-8_26.

[19] N. Halbwachs, Y.-É. Proy, P. Raymond, Verification of linear hybrid systems by means of convex approximations, in: B. Le Charlier (Ed.),

Figure A.5: MoULDyS Logo.

SAS, Vol. 864 of Lecture Notes in Computer Science, Springer, 1994, pp. 223–237. doi:10.1007/3-540-58485-4_43.

[20] A. Becchi, E. Zaffanella, Revisiting polyhedral analysis for hybrid systems, in: B. E. Chang (Ed.), SAS, Vol. 11822 of Lecture Notes in Computer Science, Springer, 2019, pp. 183–202. doi:10.1007/978-3-030-32304-2_10.

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

[22] S. Mitsch, A. Platzer, Verified runtime validation for partially observable hybrid systems, Tech. rep. (2018). arXiv:1811.06502.
URL http://arxiv.org/abs/1811.06502

[23] A. Platzer, The complete proof theory of hybrid systems, in: LICS, IEEE Computer Society, 2012, pp. 541–550. doi:10.1109/LICS.2012.64.

Appendix A. MoULDyS: A Monitoring Tool for Autonomous Systems

The various details of MoULDyS are given as follows:

Logo The tool logo is given in Fig. A.5.

Webpage The tool webpage can be found here: https://www.sites.google.com/view/mouldys.

Code `MoULDyS` is an open-source tool under the `gpl-3.0` license. The code can be found in a public GitHub repository: `https://github.com/bineet-coderep/MoULDyS/releases/tag/v1.1`.

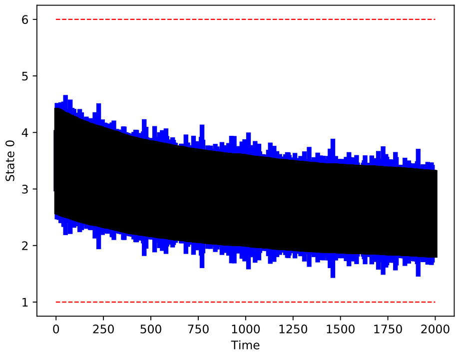
Installation Guide The installation guide is available in [2].

User Guide The user guide is available in [3].

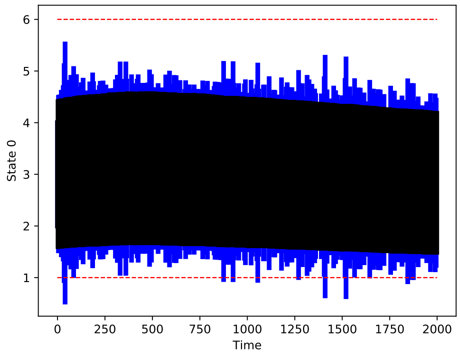
Result Recreation Guide Guide A prototype version of `MoULDyS` was used to perform the experiments in [9]. The detailed steps to recreate the results in [9], through easy-to-use scripts, are available in [14].

Appendix B. Recreating Experimental Results

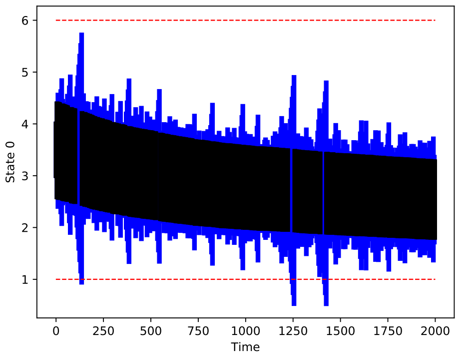
The results to be recreated for the Anesthesia case study are given in Figs. B.6 and B.7. The results to be recreated for the ACC case study are given in Figs. B.8 and B.9.



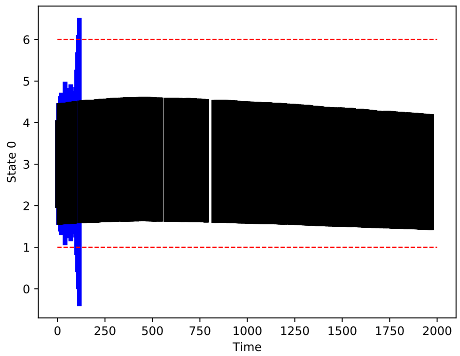
(a) Monitoring with frequent samples, and low uncertainty



(b) Monitoring with frequent samples, and high uncertainty

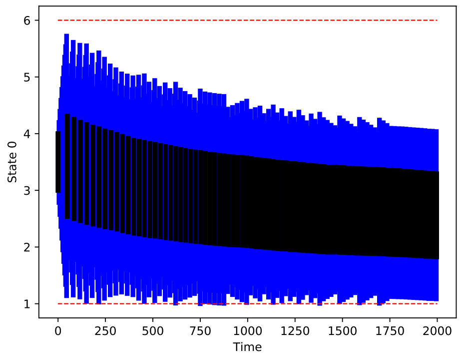


(c) Monitoring with sporadic samples, and low uncertainty

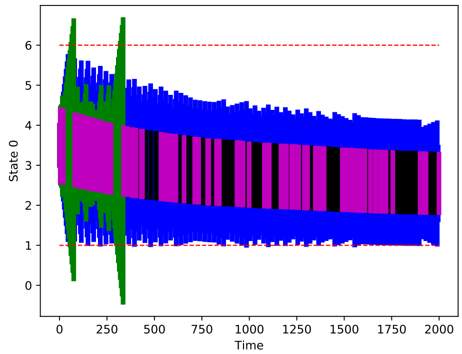


(d) Monitoring with sporadic samples, and high uncertainty

Figure B.6: ***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 **Figure 1** and **Figure 4** (Figs. B.6b and B.6c) 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. B.6d. These plots are a stochastic recreation of [9, Figure 3].

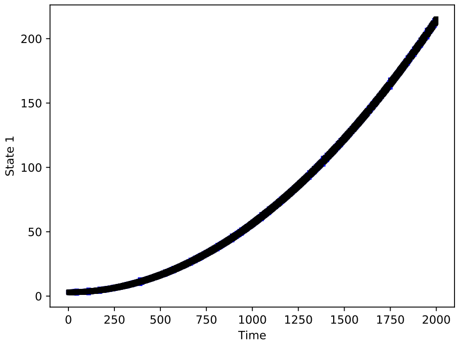


(a) **Online Monitoring**

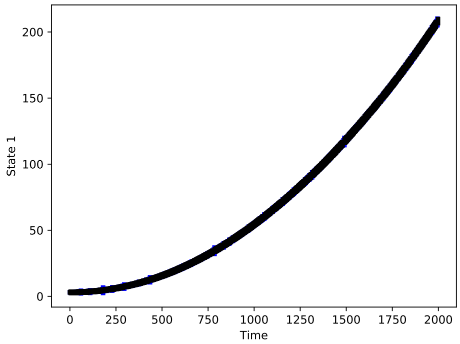


(b) **Compare Online and Offline Monitoring**

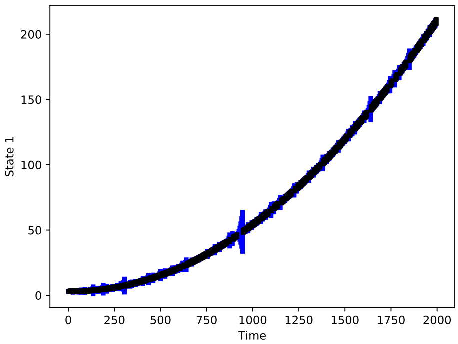
Figure B.7: ***Online Monitoring (Anesthesia).*** We plot the change in concentration level of c_p with time. The blue regions are the reachable sets showing the over-approximate reachable sets as computed by the online monitoring, the black regions are the samples generated when the logging system was triggered by the online monitoring algorithm, and the red dotted line represents safe concentration levels. *Online Monitoring (Fig. B.7a):* We apply our online monitoring to the anesthesia model. *Compare (Fig. B.7b):* 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. These plots are a stochastic recreation of [9, Figure 4].



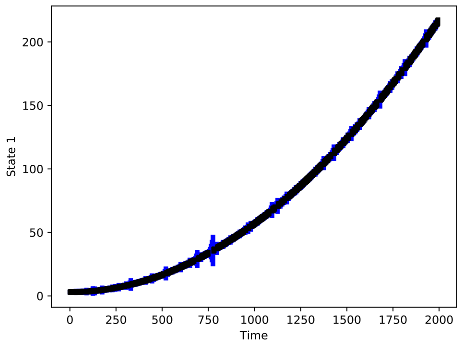
(a) Monitoring with frequent samples, and low uncertainty



(b) Monitoring with frequent samples, and high uncertainty

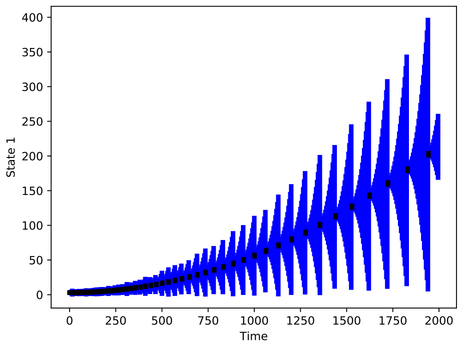


(c) Monitoring with sporadic samples, and low uncertainty

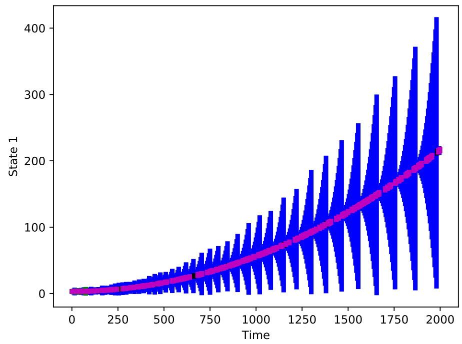


(d) Monitoring with sporadic samples, and high uncertainty

Figure B.8: ***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. These plots are a stochastic recreation of [9, Figure 5]



(a) **Online Monitoring**



(b) **Compare Online and Offline Monitoring**

Figure B.9: ***Online Monitoring (ACC).*** We plot the change in distance between two vehicle h with time. The color coding is same as Fig. B.7. *Online Monitoring (Fig. B.9a)*: We apply our online monitoring to the ACC model. *Compare (Fig. B.9b)*: We compare our online and offline algorithms. These plots are a stochastic recreation of [9, Figure 6]