Towards Machine Learning Enhanced LTL Monitoring

Luca Geatti[1], Angelo Montanari[1] and Nicola Saccomanno[1]

[1]*University of Udine, Italy*

Abstract

In this work, we outline an extension of a recently proposed framework for failure detection that additionally supports the detection of *anomalies* and *drops of performance* of a given system. The extended framework is based on a tight integration of *monitoring* with *unsupervised learning* techniques, that are used to generate formulas able to capture possible deviations from the normal behaviour of the system or early signs of degradation phenomena. Other improvements to the framework are proposed like, for instance, the use of canonical forms for the safety and cosafety (monitorable) fragments of temporal logics and the support for assumption-based runtime verification.

Keywords

Runtime verification, Monitoring, Anomaly detection, Machine learning, Interpretability

1. Introduction

Monitoring [1, 2] is a runtime verification technique [3] for the formal analysis of systems that checks a finite prefix of the current execution (*trace*) of the *system under scrutiny* to detect failures or successes expressed by means of temporal formulas. A crucial feature is that the verdict of a monitoring algorithm is *irrevocable*: once a failure (resp., a success) is detected, all continuations of the execution of the system are guaranteed to be failures (resp., successes).

Monitoring is a lightweight verification technique as it considers only the current trace execution. Thus, monitoring algorithms are usually more efficient than *model checking* ones [4], where *all* system's traces are exhaustively analyzed; in addition, the former run on the system/implementation without the need of a model, avoiding modeling errors caused by wrong or even wrong approximation of reality. Monitoring typically consists of the following steps: (1) the bad and good behaviors that will be later checked against the system's traces are specified by means of a temporal logic formulas; common choices are Linear Temporal Logic (LTL) [5] and Signal Temporal Logic (STL) [6]; (2) from each temporal formula, a *monitor* is built which typically is a deterministic finite automaton (DFA) equivalent to the initial formula; (3) the monitor is used for analysing the system. This can be done either in an *offline* (past log of the system analysed by the monitor) or in an *online* fashion (monitor paired to the running system).

The complexity of modern systems makes specifying all relevant properties for monitoring system failures challenging. Moreover, even minor system changes can introduce unforeseen

bugs. To overcome these limitations, a framework has been recently proposed with the objective of automatically discovering new relevant properties to be used to detect system's failures by means of a monitoring algorithm [7]. Specifically, machine learning is relied upon to automatically synthesize, in an iterative fashion, new relevant properties, starting from the verdicts of the monitors built from a pool of already considered formulas. This pairing of monitoring with machine learning proved itself to be very effective, making it possible to learn properties that characterize failures and their prelude increasingly in advance. Most importantly, being (temporal) formulas, the new properties that are discovered are inherently *interpretable*: this allows the system experts to easily understand what characterizes each failure event even in the case of properties automatically synthesized by the framework.

In this contribution, we conceptually extend the framework introduced in [7] along the following lines: (i) the new version of the framework must deal with *anomalies* and *drops of performance* instead of only failure events; (ii) being failures treated as catastrophic/trace-terminating while anomalies and drops of performance are not, the framework shall rely on *unsupervised and self-supervised learning* techniques; (iii) the framework shall use specific *canonical forms* of the *safety* and the *cosafety* fragments of temporal logics so to generate only properties that are *monitorable* and accommodate *unbounded* fragments of temporal logics; and, (iv) the framework must be *modular* with respect to the specification language and the backends used for monitoring and learning tasks, while preserving its key peculiarities, such as, for instance, the *interpretability* of the results.

The structure of this extended abstract is as follows: Section 2 gives a short account of the original framework, Section 3 describes the proposed improvements, and Section 4 concludes by outlining potential future directions and open problems.
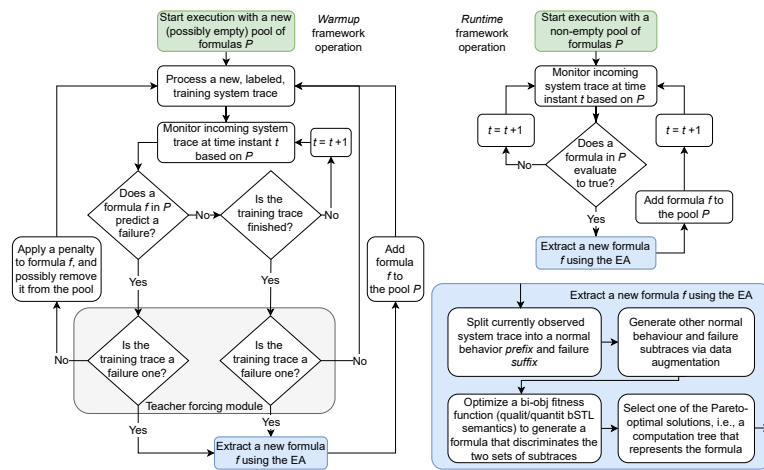
2. The original framework

Figure 1: High level functioning of the framework.

The original framework [7], depicted in Figure 1, works in an *online* fashion employing the `rtamt` monitoring tool [8].

We distinguish two execution phases: an optional *warmup* phase and a *runtime* phase. In the first one, the (possibly empty) monitoring pool $\mathcal{P}$ is populated with a set of formulas encoding bad behaviors, following a *teacher forcing*-like approach on supervised training data. In the second one, the framework monitors the system, starting with a non-empty pool $\mathcal{P}$.

During both phases, $\mathcal{P}$ is iteratively refined by (i) adding new formulas which are able to predict bad behaviors earlier and with increased reliability and coverage, and (ii) removing formulas that are ill-behaved or redundant. In addition to this automatic refinement process, at any time, domain experts can, in principle, make changes to the pool $\mathcal{P}$, e.g., by manually specifying a new formula encoding a bad behavior.

The core of the formula extraction process is a multi-objective evolutionary algorithm (EA), designed to perform a genetic programming (GP) task. Each time a formula in $\mathcal{P}$ triggers, the currently observed system subtrace is passed to the EA, which splits it into a normal behavior *prefix* and failure *suffix*. Then, after a data augmentation phase, the formula that best discriminates between good and failure behaviours is generated.

3. The new framework

In this section, we discuss the distinctive features of the extended framework we are developing.

Anomaly detection. *Anomalies* are events that deviate from the nominal system's behavior, but are not catastrophic. Their detection is of utmost importance since it can point out a possible onset of a catastrophic event (*e.g.*, a failure) to the system expert, who can then take necessary actions. They are thus of interest in the context of *predictive maintenance* [9, 10]. It is worth highlighting that the concept of anomaly is strictly more general and complex than that of failure. In fact, anomalies can be unbounded in the future, can occur several times in a trace with a different duration, can be of different types, and apply to domains beyond predictive maintenance (e.g., sleep apnea in healthcare [11]). The forthcoming version of the framework, in addition to failure detection, has to be able to detect anomalies as well as drops of performance.

Unsupervised and self-supervised learning backends. The original version of the framework relies on a supervised learning paradigm: a dataset of system traces labeled as failure or good behaviour ones is used to guide the first (so-called *warmup*) stage of formula extraction, following a teacher-forcing approach borrowed from the deep learning realm. The choice of such a learning paradigm is justified by the fact that failures are terminating event, thus, always detectable. The new version of the framework has to work also in a self-supervised fashion to deal with anomalies. Characterizing *a priori* anomalies in modern complex systems is impractical, given the multitude of heterogeneous sensors to be considered and the fact that systems evolve continuously over time, leading to previously unobserved anomalies. This scenario implies that configuring the problem as supervised, i.e., assuming that one may have a complete and exhaustive dataset of labelled anomalies, would be unrealistic and unfeasible. To solve such an issue, we plan to use deep learning based state-of-the-art approaches capable to perform self-supervised anomaly detection [12] as a source of supervision to the algorithm in charge of learning temporal formulas. It is worth noticing that interpretability is preserved, since the output of the learning step would still be a formula.

Specification language. As for the specification language to be used both to encode and to (automatically) synthesize new properties, we plan to resort to two fragments of Linear Temporal Logic with Past (LTL+P), namely, the *safety* and *cosafety* fragments [13]. In particular, we will focus on their *canonical forms* $\mathsf{G}(\mathsf{pLTL})$ and $\mathsf{F}(\mathsf{pLTL})$ [14]. This has a number of advantages:

- it avoids the specification and synthesis of formulas that are *non-monitorable*, since all

formulas of G(pLTL) and F(pLTL) belong to the monitorable class [15];

- in contrast to the *bounded fragment* used in the previous version of the framework, formulas of G(pLTL) and F(pLTL) can constrain an arbitrary time window of a trace;

- from formulas of G(pLTL) and F(pLTL), it is possible to generate *deterministic symbolic automata* that can be used as monitors and that, in the average case, are exponentially more succinct than classical monitors [16];

- the usage of these fragments can shrink the search space of the learning algorithms, leading to a faster generation of (better) formulas, acting as an inductive bias [17, 18].

Assumption-based runtime verification. Recently, *Assumption-Based Runtime Verification* (ABRV, [19]) has been introduced as a variant of standard monitoring to deal with systems that are only *partially observable*. While classical monitoring restricts itself to observable parts of the system and treat the non-observable ones as *black boxes*, ABRV exploits the fact that in practice one always knows something about the internal (non-observable) parts of the system in form of *assumptions* that the domain expert can specify before monitoring. As shown in [19], ABRV has the advantage to reach conclusive verdicts with shorter trace's prefixes. It is thus promising to include the assumption-based variant of monitoring in the framework.

Modularity. We require the framework to be *modular* in at least the following dimensions: (i) the specification language; (ii) the backend implementing the monitoring algorithm; (iii) the backend for the learning of new properties. As for the specification language, the framework shall support different temporal logics including LTL, STL, and ITL (Interval Temporal Logic), accommodating for both the *qualitative* semantics (for tasks like failure detection) and the *quantitative* one (for tasks like anomaly detection), where appropriate. Modularity of the learning backend allows one to seamlessly move across learning paradigms and tasks (e.g., failure and anomaly detection). Moreover, it allows one to consider a different solution than GP for formula extraction (which is limited by bloat, huge search space, tree based formula representation, *etc.*), like the integration with reinforcement learning or generative AI [20]; alternatively, formulas can be represented as graphs, enabling the usage of Graph Neural Networks [21]. Finally, a modular monitoring backend is important for implementing new, modern techniques that have been proposed in the literature [19, 22].

4. Conclusions, open problems, and future research directions

In this abstract, we discussed an extension of the framework proposed in [7]. Its cornerstone features are (i) interpretable failure/anomaly detection, (ii) unsupervised/self-supervised learning, (iii) theoretically sound formulation, and (iv) modularity.

Besides implementing the described features, future research directions concern: managing contradictions and tautologies in the monitoring pool, e.g., by using solutions from Inductive Logic Programming [23]; providing estimation for the remaining useful life; supporting the existence of multiple, parallel pools of formulas; and, investigating the usage of the framework in other tasks and domains, such as, for instance, detecting biases and inconsistencies in ground truth labels over temporal data in healthcare, leveraging its interpretability.

Acknowledgments

References

[1] I. Cassar, A. Francalanza, L. Aceto, A. Ingólfsdóttir, A survey of runtime monitoring instrumentation techniques, in: A. Francalanza, G. J. Pace (Eds.), Proceedings Second International Workshop on Pre- and Post-Deployment Verification Techniques, PrePost@iFM 2017, Torino, Italy, 19 September 2017, volume 254 of *EPTCS*, 2017, pp. 15–28. URL: https://doi.org/10.4204/EPTCS.254.2. doi:$\overline{T}$10.4204/EPTCS.254.2.

[2] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, K. Lehtinen, An operational guide to monitorability with applications to regular properties, Softw. Syst. Model. 20 (2021) 335–361. URL: https://doi.org/10.1007/s10270-020-00860-z. doi:$\overline{T}$10.1007/s10270-020-00860-z.

[3] M. Leucker, C. Schallhart, A brief account of runtime verification, The Journal of Logic and Algebraic Programming 78 (2009) 293–303. doi:$\overline{T}$10.1016/j.jlap.2008.08.004.

[4] E. M. Clarke, T. A. Henzinger, H. Veith, R. Bloem, Handbook of model checking, volume 10, Springer, 2018. doi:$\overline{T}$10.1007/978-3-319-10575-8.

[5] A. Pnueli, The temporal logic of programs, in: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), IEEE, 1977, pp. 46–57. doi:$\overline{T}$10.1109/SFCS.1977.32.

[6] A. Donzé, O. Maler, E. Bartocci, D. Nickovic, R. Grosu, S. A. Smolka, On temporal logic and signal processing, in: S. Chakraborty, M. Mukund (Eds.), Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings, volume 7561 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 92–106. URL: https://doi.org/10.1007/978-3-642-33386-6_9. doi:$\overline{T}$10.1007/978-3-642-33386-6_9.

[7] A. Brunello, D. D. Monica, A. Montanari, N. Saccomanno, A. Urgolo, Monitors that learn from failures: Pairing STL and genetic programming, IEEE Access 11 (2023) 57349–57364. URL: https://doi.org/10.1109/ACCESS.2023.3277620. doi:$\overline{T}$10.1109/ACCESS.2023.3277620.

[8] D. Nickovic, T. Yamaguchi, RTAMT: online robustness monitors from STL, in: D. V. Hung, O. Sokolsky (Eds.), Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings,

volume 12302 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 564–571. URL: https://doi.org/10.1007/978-3-030-59152-6_34. doi:$\overline{T}$10.1007/978-3-030-59152-6_34.

[9] R. K. Mobley, An introduction to predictive maintenance, Elsevier, 2002.

[10] T. Zonta, C. A. Da Costa, R. da Rosa Righi, M. J. de Lima, E. S. da Trindade, G. P. Li, Predictive maintenance in the industry 4.0: A systematic literature review, Computers & Industrial Engineering 150 (2020) 106889.

[11] A. Bernardini, A. Brunello, G. L. Gigli, A. Montanari, N. Saccomanno, AIOSA: an approach to the automatic identification of obstructive sleep apnea events based on deep learning, Artif. Intell. Medicine 118 (2021) 102133. URL: https://doi.org/10.1016/j.artmed.2021.102133. doi:$\overline{T}$10.1016/j.artmed.2021.102133.

[12] H. Hojjati, T. K. K. Ho, N. Armanfard, Self-supervised anomaly detection: A survey and outlook, CoRR abs/2205.05173 (2022). URL: https://doi.org/10.48550/arXiv.2205.05173. doi:$\overline{T}$10.48550/arXiv.2205.05173. `arXiv:2205.05173`.

[13] A. P. Sistla, On characterization of safety and liveness properties in temporal logic, in: Proceedings of the fourth annual ACM symposium on Principles of distributed computing, 1985, pp. 39–48. doi:$\overline{T}$10.1145/323596.323600.

[14] E. Y. Chang, Z. Manna, A. Pnueli, Characterization of temporal property classes, in: W. Kuich (Ed.), Proceedings of the 19th International Colloquium on Automata, Languages and Programming, volume 623 of *Lecture Notes in Computer Science*, Springer, 1992, pp. 474–486. doi:$\overline{T}$10.1007/3-540-55719-9_97.

[15] A. Bauer, M. Leucker, C. Schallhart, Runtime verification for ltl and tltl, ACM Transactions on Software Engineering and Methodology (TOSEM) 20 (2011) 1–64. doi:$\overline{T}$10.1007/BF01068590.

[16] A. Cimatti, L. Geatti, N. Gigante, A. Montanari, S. Tonetta, Extended bounded response LTL: a new safety fragment for efficient reactive synthesis, Formal Methods in System Design (2021) 1–49 (published online on November 18, 2021, doi: 10.1007/s10703−021−00383−3).

[17] D. F. Gordon, M. Desjardins, Evaluation and selection of biases in machine learning, Machine learning 20 (1995) 5–22.

[18] T. M. Mitchell, The need for biases in learning generalizations (1980).

[19] A. Cimatti, C. Tian, S. Tonetta, Assumption-based runtime verification, Formal Methods Syst. Des. 60 (2022) 277–324. URL: https://doi.org/10.1007/s10703-023-00416-z. doi:$\overline{T}$10.1007/s10703-023-00416-z.

[20] S. Holt, Z. Qian, M. van der Schaar, Deep generative symbolic regression, in: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023, OpenReview.net, 2023. URL: https://openreview.net/pdf?id=o7koEEMA1bR.

[21] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net, 2017. URL: https://openreview.net/forum?id=SJU4ayYgl.

[22] X. Qin, J. V. Deshmukh, Clairvoyant monitoring for signal temporal logic, in: N. Bertrand, N. Jansen (Eds.), Formal Modeling and Analysis of Timed Systems - 18th International Conference, FORMATS 2020, Vienna, Austria, September 1-3, 2020, Proceedings, volume 12288 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 178–195. URL: https://doi.org/10.1007/978-3-030-57628-8_11. doi:$\overline{T}$10.1007/978-3-030-57628-8_11.

[23] L. D. Raedt, Inductive logic programming, in: C. Sammut, G. I. Webb (Eds.), Encyclopedia of Machine Learning and Data Mining, Springer, 2017, pp. 648–656. URL: https://doi.org/10.1007/978-1-4899-7687-1_135. doi:$\overline{T}$10.1007/978-1-4899-7687-1_135.