Learning to Monitor: a Novel Framework for Online System Verification[*]

Andrea Brunello[†] , **Dario Della Monica** , **Angelo Montanari** , **Andrea Urgolo**
University of Udine, Italy
{andrea.brunello,dario.dellamonica,angelo.montanari}@uniud.it, urgolo.andrea@spes.uniud.it

Abstract

In several domains, the execution of systems is associated with the generation of continuous streams of data. The streams may contain important telemetry information, which can be used to perform tasks like predictive maintenance and preemptive failure detection, in order to issue early warnings. In critical contexts, formal methods have been recognized as an effective approach to ensure the correct behaviour of a system. However, they have at least two significant weaknesses: (i) a complete, hand-made specification of all the properties that have to be guaranteed during the execution of the system turns out to be often out of reach when complex systems have to be handled and, for the same complexity reasons, (ii) it may be difficult to derive a complete model of the system against which to check the properties of interest. To overcome these limitations, some approaches that complement formal verification with model-based testing and monitoring have been recently proposed. In this paper, we argue for the opportunity of pairing monitoring with machine learning techniques in order to improve its ability of detecting critical system behaviours in an on-line, data streaming setting.

1 Introduction

Typically, during its execution, a system generates several streams of data, which may contain important telemetry information. As an example, this is the case with logs produced by web servers, smart sensors, or machinery in modern industrial plants. System behaviours may be arbitrarily convoluted, as they can be the result of the interaction among several components, and between these components and the surrounding environment.

In such a complex setting, formal methods can be exploited as effective tools for the automatic verification of software and hardware systems, a task which is of paramount importance in many critical domains. However, the inherent complexity of system's components and of their interactions make it very difficult (and sometimes impossible) to specify in advance all the relevant properties that have to be guaranteed (or, dually, avoided) during their execution. In addition, the definition of a complete model of the system against which to check the properties of interest may also be out of reach.

To overcome these limitations, some approaches that complement formal verification with model-based testing and monitoring have been recently proposed in the literature (see, for instance, (Cassar et al. 2017; Gerhold, Hartmanns, and Stoelinga 2019)).

In this work, we focus on *monitoring* (Leucker and Schallhart 2009), a runtime verification technique which is gaining more and more attention within the formal verification community. The key feature of monitoring is that it allows one to detect satisfaction or violation of a property (typically expressed by some temporal logic formula) by analyzing a single run of the system. This makes such a technique naturally applicable to data streaming contexts.

In the remainder, we outline a novel framework for online system verification that integrates monitoring with machine learning and can be applied in preemptive failure detection and predictive maintenance tasks in data streaming contexts[1]. As we shall see, the proposed approach allows a domain expert to start with the specification of the most important and natural properties to monitor. Then, the framework autonomously discovers new relevant properties by means of an iterative refinement process, becoming capable, over time, of identifying undesired behaviours in advance, with a considerably higher level of detail and coverage than the original specification.

2 Online System Verification

As it is emerging from the most recent literature in the field, machine and statistical learning solutions can be successfully combined with formal methods techniques to deal with complex real-world problems (Jansen et al. 2018). In the following, we briefly outline a possible integration of monitoring and machine learning, developing a framework that can be exploited for online system verification.

2.1 The Framework

The framework operation consists of five main phases.

Specification of the initial set of properties. Domain experts are asked to specify the most significant (monitorable)

[*]The authors acknowledge the partial support by the Italian INdAM-GNCS project *Ragionamento Strategico e Sintesi Automatica di Sistemi Multi-Agente*.

[†]Speaker.

[1]For more details, see (Brunello, Della Monica, and Montanari 2019) and references therein.

properties that the system under consideration should exhibit. These properties are then formalized in a suitable temporal logic and a monitor that checks them against incoming execution traces is synthesized.

Monitoring of system properties. The monitor checks whether the system satisfies/violates the specified properties during its execution.

Detection of a failure. Traces for which the monitor reaches a verdict of failure are collected. These are considered to be *failure* traces. In addition, traces generated by the system during previous normal behaviour are extracted, and considered to be *normal* traces. Of course, the length of the time window that is used to extract failure traces depends on the specific application domain, and it must be carefully chosen according to the results of a dedicated tuning phase, possibly with the help of domain experts.

Mining of the relevant behaviour patterns. Failure and normal traces are put together to generate a dataset for supervised machine learning. Each instance is characterized by a (possibly multivariate) trace that can contain numerical (as in the case of a temperature signal) or categorical (this is the case, for instance, with a sequence of system calls made in a Unix system) values. Moreover, each instance is labeled with a binary class, that can be either *failure* or *normal* behaviour. Traces are first converted into timelines (see, e.g., (Sciavicco, Stan, and Vaccari 2019)). Then, pattern mining algorithms are run on the dataset of labeled timelines, with the goal of extracting the (temporal) logic formulas that best characterize and discriminate between the two classes (following, for instance, the approaches described in (Bresolin et al. 2018; Brunello, Sciavicco, and Stan 2019)).

Extension of the pool of properties. The (temporal) logic formulas extracted during the mining phase are added to the monitoring pool of properties, and the process restarts from the monitoring phase.

The proposed framework works in an iterative way, which we may refer to as its *online* phase, in which incoming traces are considered. It starts from a set of basic properties, and new ones are then added over time. The discovered logical properties are closely related to the original ones and, in principle, they should allow the system to discover anomalous behaviours earlier and with ever increasing accuracy and coverage.

On the basis of the organization of the above five steps, we can identify different framework phases and learning modes.

2.2 Warmup and Online Execution Phases

Sometimes, data pertaining to past system failures may be available, or approximate data may be generated by means of simulations. In that case, it makes sense to exploit these pieces of information to perform monitor learning even before its *online* phase. To do that, intuitively, it is sufficient to mimic the continual arrival of the available traces, and to iteratively follow the five steps which we described. Thanks to this initial *warmup* phase, the framework can then deal with the subsequent *online* phase starting with an already-extended pool of properties.

2.3 Semi-supervised and Unsupervised Learning

Let us focus on the task of preemptive machinery fault detection. According to the first step of the framework, a domain expert may be required to specify an initial set of properties which should be monitored against the execution of the system, thus acting, in her/his vision, as fault early warnings. Since there is, at first, a human intervention, we can refer to this strategy as a kind of *semi-supervised* learning.

Nevertheless, domain expert knowledge may sometimes not be available. In such a case, the monitor is initialized with just a single, trivial property, that is, "the machinery is in operation". Then, once a failure is detected (indeed not in a preemptive way), the framework may proceed with the usual steps in order to detect some properties that may help it in forecasting the fault before it actually happens. Since in this operation mode there is no human intervention (except for the trivial, initial, "machinery in operation" property), we can refer to it as a kind of *unsupervised* learning.

References

Bresolin, D.; Cominato, E.; Gnani, S.; Muñoz-Velasco, E.; and Sciavicco, G. 2018. Extracting interval temporal logic rules: A first approach. In *Proceedings of the 25th International Symposium on Temporal Representation and Reasoning*, volume 120 of *LIPIcs*, 7:1–7:15.

Brunello, A.; Della Monica, D.; and Montanari, A. 2019. Pairing monitoring with machine learning for smart system verification and predictive maintenance. In *Proceedings of the 1st Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis*, volume 2509 of *CEUR Workshop Proceedings*, 71–76.

Brunello, A.; Sciavicco, G.; and Stan, I. E. 2019. Interval temporal logic decision tree learning. In *Proceedings of the 16th European Conference on Logics in Artificial Intelligence*, volume 11468 of *Lecture Notes in Computer Science*, 778–793.

Cassar, I.; Francalanza, A.; Aceto, L.; and Ingólfsdóttir, A. 2017. A survey of runtime monitoring instrumentation techniques. In *Proceedings of the 2nd International Workshop on Pre- Post-Deployment Verification Techniques*, 15–28.

Gerhold, M.; Hartmanns, A.; and Stoelinga, M. 2019. Model-based testing of stochastically timed systems. *Innovations in Systems and Software Engineering* 15(3-4):207–233.

Jansen, N.; Katoen, J.-P.; Kohli, P.; and Kretinsky, J. 2018. Machine learning and model checking join forces. *Dagstuhl Reports* 8(3):74–93.

Leucker, M., and Schallhart, C. 2009. A brief account of Runtime Verification. *Journal of Logical and Algebraic Methods in Programming* 78(5):293–303.

Sciavicco, G.; Stan, I. E.; and Vaccari, A. 2019. Towards a general method for logical rule extraction from time series. In *Proceedings of the 8th International Work-Conference on the Interplay Between Natural and Artificial Computation,*, volume 11487 of *Lecture Notes in Computer Science*, 3–12.