Pairing monitoring with machine learning for smart system verification and predictive maintenance

Andrea Brunello[1], Dario Della Monica[2], and Angelo Montanari[3]

[1,2,3]University of Udine, Italy
[1]andrea.brunello@uniud.it
[2]dario.dellamonica@uniud.it
[3]angelo.montanari@uniud.it

Abstract

Over the last decades, the advancements in microelectronic technologies allowed for the embedding of complex digital sensors in several systems, ranging from home appliances to health tracking devices and industrial plant machinery. The resulting systems are, in general, quite complex, given the possible heterogeneity of their components and the non-trivial ways in which sensors may interact. In critical domains, formal methods have been employed to ensure the correct behaviour of a system. However, a complete specification of all the properties that have to be guaranteed turns out to be often out of reach, due to the inherent complexity of the system and of its interactions with the environment in which it operates. 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.

1 Introduction

Over the last decades, thanks to new microelectronic technologies, even the simplest measuring devices have progressively evolved into complex digital systems capable of performing several processing and communication operations related to the measured values (smart sensors). A smart sensor can be defined as a device that takes its input from the physical environment and uses built-in computational resources to perform some predefined functions upon detection of specific input, thus processing data before passing them on [10]. Such devices may be used to set up monitoring and control mechanisms in a variety of contexts, including smart grids [15], and they have several applications in agriculture [20], health-care [12], and industry [18]. As an example, one may think of an industrial plant, where the behaviour of all the different subsystems and components is tracked by means of dedicated smart sensors embedded in the machinery.

In such a complex environment, formal methods can be exploited to devise methods and techniques for the automated verification of software and hardware systems, a task which is of paramount importance in many critical domains. However, the inherent complexity of the systems and the non-trivial ways in

which their components may interact among themselves and with the environment, make it very difficult (and sometimes impossible) to specify in advance all the relevant properties that have to be guaranteed (or, dually, avoided). 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, [7, 11]).

In this contribution, we focus our attention on *monitoring*, a runtime verification technique which is gaining more and more attention within the formal method and verification community. We outline a novel framework for online system verification that integrates monitoring with machine learning and can be applied in anomaly detection and predictive maintenance tasks. 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.

This extended abstract is organized as follows. Section 2 provides a short introduction to the concept of system monitoring. Section 3 deals with machine learning and, specifically, with pattern mining techniques that may be used to analyze system traces to extract relevant properties. In section 4, we envision a framework that combines system monitoring and machine learning. A short recap of the distinctive features of the proposed approach concludes the extended abstract.

2 System monitoring

Monitoring [16] is a runtime verification technique that is gaining much interest in the realm of formal methods for automated system verification.

Classic, static techniques, such as, for instance, model checking [8], perform an exhaustive analysis of the system: satisfaction or violation of a property (typically expressed by some temporal logic formula) by the system is established by exploring all possible behaviours. On the contrary, monitoring allows one to detect satisfaction or violation of a property by analyzing a single behaviour, called *execution trace* or *run*, of the system. It is therefore a lightweight technique, but the gain in efficiency is paid in terms of expressivity: monitorable properties are a subset of the class of specifications expressible in temporal formalisms commonly used for automated verification.

Even though monitoring has been mostly studied in the setting of linear time temporal logics [13], a notable effort has been devoted to the investigation of monitoring branching-time properties (see, for instance, [9]). A comparison of the two settings can be found in [1].

The basic principles on which monitors (and the theory of monitorability) are built are the ability of reaching a verdict by just observing a finite prefix of a single execution trace (a *finite trace*) and the fact that, once a verdict is reached, it is irrevocable, as it is a guarantee that the system satisfies or violates the property, independently from all the other possible (unobserved) behaviours it might exhibit. This latter feature ensures the *soundness* of a monitor. We introduce and discuss the notion of *completeness* of a monitor below.

We say that a property is *positively monitorable* if every system satisfying it features a finite trace that witnesses the satisfaction; conversely, we say that a property is *negatively monitorable* if every system violating it features a finite trace that witnesses the violation. A *monitorable* property is a property that is either positively or negatively monitorable.

Safety properties are negatively monitorable, as their violation is witnessed by a finite trace. Dually, co-safety properties are positively monitorable. On the other hand, there exist properties like

it is possible to reach a `success` *state, but it is not possible to reach it in less than 3 steps*

which are neither positively nor negatively monitorable. A monitor observing a system execution trace that is compatible with the property, e.g., a `success` state is reached at the third execution step, indeed, cannot issue a satisfaction verdict, thus stating that the property is fulfilled by every run, as there might be a run along which a `success` state is reached in less than 3 steps. Moreover, a system might violate the property by never reaching a `success` state at all; in this case, no finite trace witnesses the violation, as the `success` state might be reached at the next step, or through a completely different run.

Instead, the property

it is not possible to reach a `success` *state in less than 3 steps*

is negatively monitorable, as every system violating it features a finite trace that reaches a `success` state in less than 3 steps.

Given a positively (resp., negatively) monitorable property P, we say that a monitor is *satisfaction-complete* (resp., *violation-complete*) for P if every system satisfying P features a finite trace that is accepted (resp., rejected) by the monitor.

It is clear that sound and complete monitors only exist for monitorable properties. Thus, characterizing the set of monitorable properties is one of the major challenges in the field. However, some work has also been done in the attempt of weakening the notion of monitorability (see [2] for an account of such a research direction) or of extending the realm of applicability of such a technique to specifications that are deemed not amenable to being runtime verified [9].

3 Pattern mining and machine learning

Machine learning (ML) is an area of Artificial Intelligence (AI) that focuses on algorithms and statistical models that have the ability to automatically learn (and improve their behaviour) from experience without being explicitly programmed. In particular, in *supervised* machine learning, the goal is that of building a model capable of mapping a given input to a desired output. Depending on the kind of output to generate, it is possible to distinguish between *regression* and *classification* tasks, characterized by numerical and categorical (classes) values, respectively.

In order to learn a model, a machine learning algorithm undergoes a so-called *training* phase, during which several examples (or instances) of the concept that has to be learnt are taken into consideration. For each considered instance, both the input and the output variables are known. Once the model has been built, it can be applied to new instances, for which only the input variables are known, in order to predict the output values.

Among supervised learning tasks, in this work we are mainly interested in *pattern mining*. In such a setting, the input consists of a sequence of numerical (time series) or categorical (e.g., sequences of system states) values, whereas the output is, typically, categorical. The goal is to extract a set of relevant patterns from the input sequences, that characterize (and make it possible to distinguish among) instances belonging to different classes.

Patterns can be of different kinds. For lists of categorical values, *frequent patterns* have been investigated (see, for instance, [3, 19, 23]). As for time series, *shapelets* have been defined as contiguous, arbitrary-length time series subsequences which are in some sense maximally representative of a class [22]. A decision tree algorithm which is able of dealing with both kinds of data at the same time has been recently proposed in the literature [4].

As for a more formal characterization of a pattern, the task of learning temporal logic formulas from examples is attracting an increasing interest. For instance, a decision tree learning algorithm which is capable of generating interval temporal logic formulas from sequences of categorical values, in order to perform a classification task, is described in [5], while a technique to learn general, unrestricted linear temporal logic (LTL) formulas from a set of examples in the form of infinite, ultimately periodic words, without relying on user-defined templates, is illustrated in [17]. Variants and extensions of the latter result are considered in [6, 21]. More precisely, the case of LTL interpreted over finite traces is dealt with in [6], while parametric linear temporal logic (pLTL) is analyzed in [21].

4 Integrating system monitoring and machine learning

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 [14]. One challenging scenario is that of failure detection and predictive maintenance. As we have already pointed out, formal methods allow one to synthesize monitors to check whether some good properties (safety properties) are violated by a system during its operation and to detect those system's behaviours

(traces) that violate them. In the following, we briefly outline a possible integration of monitoring and machine learning, that can be decomposed into five main phases.

Specification of the initial set of properties. Domain experts are asked to specify the most significant 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 execution traces is synthesized (obviously, monitorability of the relevant formulas is a critical issue here).

Monitoring of system properties. The monitor checks whether the system satisfies/violates the considered properties during its execution (monitoring).

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 trace that can be 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). Moreover, each instance is labeled with a binary class, that can be either *failure* or *normal* behaviour. Pattern mining algorithms are run on the dataset, with the goal of extracting the (temporal) logic formulas that best characterize and discriminate between the two classes.

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

As it can be observed, the proposed framework works in an iterative way. 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.

5 Conclusions

Thanks to the new microelectronic technology advancements, measuring devices can now be easily transformed into complex digital systems capable of processing data, and of interacting with one another and with the surrounding environment. As a result, their emerging behaviour is typically non trivial, and difficult to formalize into a set of logical properties to be used for standard system verification, even for a pool of domain experts. For such a reason, approaches that complement formal verification with model-based testing and monitoring have been recently investigated in the literature.

In this work, we outlined a possible solution to such a problem which integrates machine learning and monitoring techniques. The proposed formal verification framework starts with the runtime verification of the most important and natural properties that should be guaranteed during the execution of the analyzed system. Then, it autonomously discovers new ones in an iterative way, becoming capable of identifying unwanted behaviours earlier, and with ever higher detail and coverage than the original specification. In the long run, the verification tool is expected to gain the ability of performing advanced tasks such as self-diagnosis, predictive maintenance, adaptation, and robustness enhancement.

A practical implementation of the system introduced here in the context of a real-world scenario represents a natural development of the present work.

References

[1] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen. Adventures in monitorability: from branching to linear time and back again. *Proceedings of the ACM on Programming Languages*, 3(POPL):52:1–52:29, 2019.

[2] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen. An operational guide to monitorability. In *Proceedings of the 17th International Conference on Software Engineering and Formal Methods (SEFM)*, volume 11724 of *Lecture Notes in Computer Science*, pages 433–453. Springer, 2019.

[3] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 429–435, 2002.

[4] A. Brunello, E. Marzano, A. Montanari, and G. Sciavicco. J48SS: A novel decision tree approach for the handling of sequential and time series data. *Computers*, 8(1):21, 2019.

[5] A. Brunello, G. Sciavicco, and I. E. Stan. Interval temporal logic decision tree learning. In *Proceedings of the 16th European Conference on Logics in Artificial Intelligence (JELIA)*, pages 778–793, 2019.

[6] A. Camacho and S. A. McIlraith. Learning interpretable models expressed in linear temporal logic. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS), to be published*, 2019.

[7] I. Cassar, A. Francalanza, L. Aceto, and A. Ingólfsdóttir. A survey of runtime monitoring instrumentation techniques. In *Proceedings of the 2nd International Workshop on Pre- and Post-Deployment Verification Techniques (PrePost@iFM)*, pages 15–28, 2017.

[8] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[9] A. Francalanza, L. Aceto, and A. Ingolfsdottir. Monitorability for the Hennessy–Milner Logic with Recursion. *Formal Methods in System Design*, pages 1–30, 2017.

[10] R. Frank. *Understanding smart sensors*. Artech House, 2013.

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

[12] P. Gupta, D. Agrawal, J. Chhabra, and P. Dhir. IoT based smart healthcare kit. In *Proceedings of the International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)*, pages 237–242. IEEE, 2016.

[13] K. Havelund and D. Peled. Runtime verification: From propositional to first-order temporal logic. In *Proceedings of the 18th International Conference on Runtime Verification (RV)*, volume 11237 of *Lecture Notes in Computer Science*, pages 90–112. Springer, 2018.

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

[15] M. Jaradat, M. Jarrah, A. Bousselham, Y. Jararweh, and M. Al-Ayyoub. The internet of energy: smart sensor networks and big data management for smart grid. *Procedia Computer Science*, 56:592–597, 2015.

[16] M. Leucker and C. Schallhart. A brief account of Runtime Verification. *JLAP*, 78(5):293–303, 2009.

[17] D. Neider and I. Gavran. Learning linear temporal properties. In *Proceedings of the 18th Conference on Formal Methods in Computer Aided Design (FMCAD)*, pages 1–10, 2018.

[18] S. Nihtianov and A. Luque. *Smart Sensors and MEMS: Intelligent Sensing Devices and Microsystems for Industrial Applications*. Woodhead Publishing, 2018.

[19] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Mining sequential patterns by pattern-growth: the prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440, 2004.

[20] G. Vellidis, M. Tucker, C. Perry, C. Kvien, and C. Bednarz. A real-time wireless smart sensor array for scheduling irrigation. *Computers and electronics in agriculture*, 61(1):44–50, 2008.

[21] Z. Xu, M. Ornik, A. A. Julius, and U. Topcu. Information-guided temporal logic inference with prior knowledge. In *Proceedings of the 2019 American Control Conference (ACC)*, pages 1891–1897, 2019.

[22] L. Ye and E. Keogh. Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 947–956. ACM, 2009.

[23] M. J. Zaki. SPADE: an efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1):31–60, 2001.