University of Udine

Learning from Failures

Machine Learning-based Monitoring for Runtime System Verification

Andrea Brunello - andrea.brunello@uniud.it
Andrea Urgolo - urgolo.andrea@spes.uniud.it

Introduction

In several domains, systems generate continuous streams of data which may contain useful telemetry information

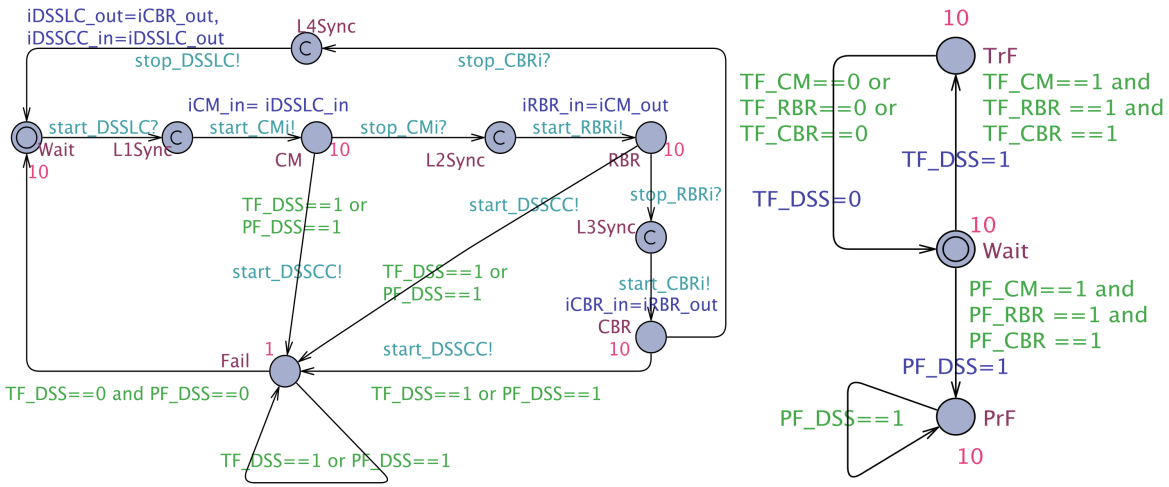
- They can be used for tasks such as predictive maintenance and preemptive failure detection
- System behaviours can be convoluted, being the result of the interaction among several components and the environment (Industry 4.0)
- Given the complexity of this setting, deep learning approaches have also been considered. Problems:
  - resulting models are hardly interpretable
  - difficulty in providing guarantees on the obtained results

Formal Methods

In critical contexts, formal methods have been recognized as an effective approach to ensure the correct behaviour of a system.

However, classical techniques, such as model checking, require a complete specification of the system and of the properties to be checked against it, in an offline fashion.

-> In some cases this may be very difficult!



A Novel Approach

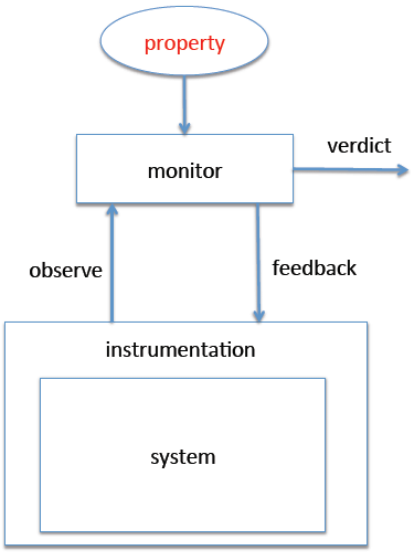
Framework that combines machine learning and monitoring to detect critical system behaviours in an on-line setting:

- system behaviour's complexity is dealt with by means of machine learning
- extracted formal properties are interpretable, so a domain expert can easily read and validate the generated model

Monitoring

Monitoring is a run-time verification technique:

- establishes satisfaction/violation of a property analyzing a finite prefix of a single behaviour (run) of the system
- lightweight technique compared to model checking
- naturally applicable to data streaming contexts



Monitoring: Monitorable Properties

When the monitor reaches a verdict, the latter is definitive.

Positively monitorable properties:

> every system satisfying it features a finite trace witnessing the satisfaction
>
> *it is possible to reach a* `success` *state*

Negatively monitorable properties:

> every system violating it features a finite trace witnessing the violation
>
> *it is not possible to reach a* `success` *state in less than 2 steps*

Not all properties are monitorable:

> *it is possible to reach a* `success` *state, but it is not possible to reach it in less than 2 steps*

Linear Temporal Logic [Pnueli 1977]

Linear Temporal Logic (LTL) is an extension of propositional Boolean logic with modalities that allow one to express temporal properties over linear structures (e.g., individual computation paths).

$$\varphi := \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 U \varphi_2 \mid X\varphi$$

$$\pi, s_i \models p \in P \Leftrightarrow V(p, s_i) = true$$
$$\pi, s_i \models \neg\alpha \iff \pi, s_i \not\models \alpha$$
$$\pi, s_i \models \alpha \wedge \beta \iff \pi, s_i \models \alpha \ \wedge \ \pi, s_i \models \beta$$
$$\pi, s_i \models X\alpha \iff \pi, s_{i+1} \models \alpha$$
$$\pi, s_i \models F\alpha \iff \exists j \geq i \ : \ \pi, s_j \models \alpha$$
$$\pi, s_i \models G\alpha \iff \forall j \geq i \ : \ \pi, s_j \models \alpha$$
$$\pi, s_i \models \alpha U \beta \iff \exists j \geq i \ : \ \pi, s_j \models \beta \ \text{e} \ \forall k \ s.t. \ i \leq k < j \ : \ \pi, s_k \models \alpha$$

LTL: Monitorable Properties

Positively monitorable properties:

every system satisfying it features a finite trace witnessing the satisfaction

it is possible to reach a `success` *state*: F `success`

Negatively monitorable properties:

every system violating it features a finite trace witnessing the violation

it is not possible to reach a `success` *state*: G `!` `success`

Not all properties are monitorable:

it is possible to reach a `success` *state and there is also a step from which is not possible to reach a* `success` *state anymore*:
F `success` `&` F(`!` G `success`)

Signal Temporal Logic [Maler et al. 2004]

Signal Temporal Logic (STL) is an extension of LTL with *real-time* and *real-valued* constraints

$$\varphi := true \mid f(\mathbf{x}) \sim c \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \, \mathbf{U}_I \, \varphi_2$$

$$\begin{aligned}
(\mathbf{x}, t) &\models f(\mathbf{x}) \sim c && \iff && f(\mathbf{x}(t)) \sim c \text{ is true} \\
(\mathbf{x}, t) &\models \neg\varphi && \iff && (\mathbf{x}, t) \not\models \varphi \\
(\mathbf{x}, t) &\models \varphi_1 \wedge \varphi_2 && \iff && (\mathbf{x}, t) \models \varphi_1 \ \wedge \ (\mathbf{x}, t) \models \varphi_2 \\
(\mathbf{x}, t) &\models \varphi_1 \, \mathbf{U}_I \, \varphi_2 && \iff && \exists t_1 \in t \oplus I : (\mathbf{x}, t_1) \models \varphi_2 \ \wedge \\
& && && \quad \forall t_2 \in [t, t_1) : (\mathbf{x}, t_2) \models \varphi_1
\end{aligned}$$

where $2\ f\ ;\ ;=g$ and $I := (a;b)j(a;b]j[a;b)j[a;b]$ with $a;b\ 2\ R^{\geq 0}$ and $a\ \ b$

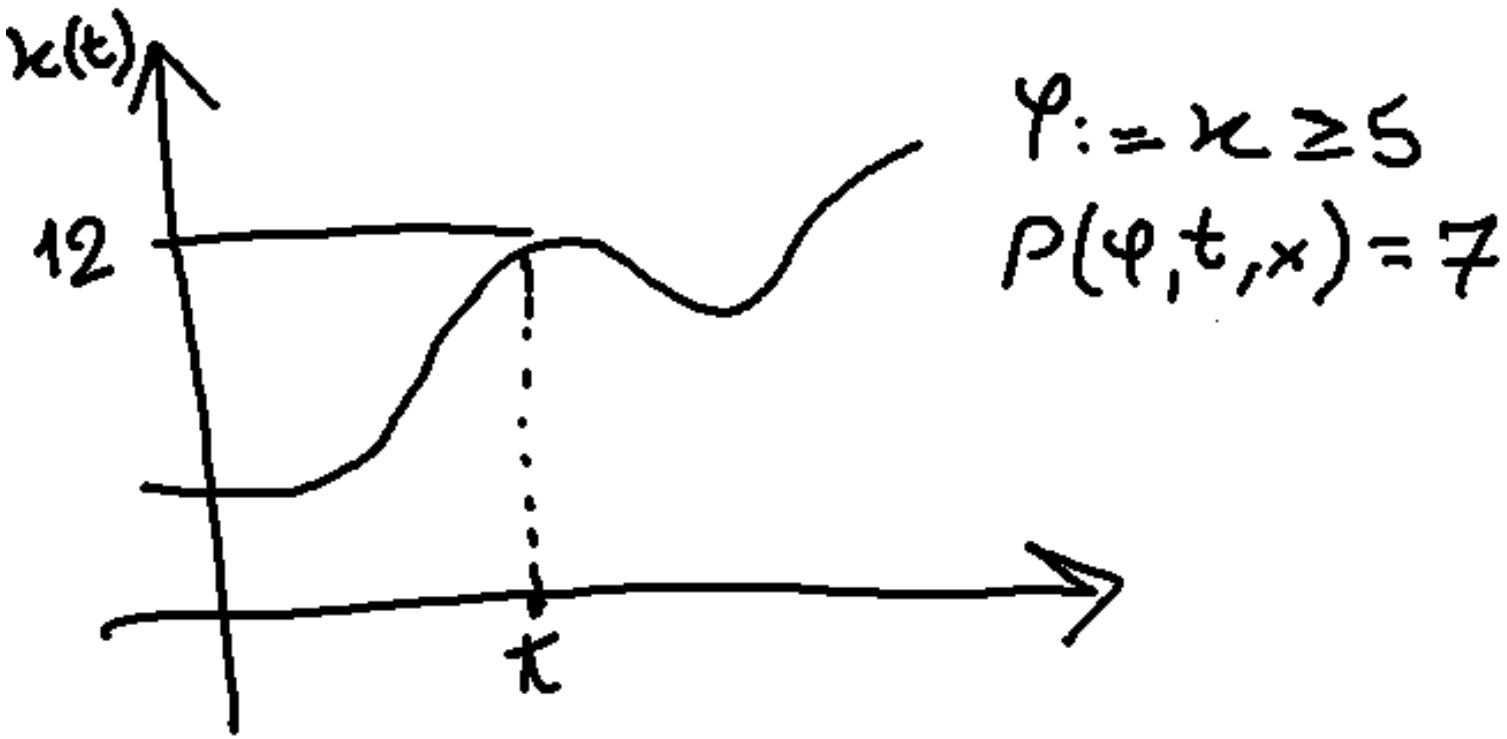
STL: Quantitative Semantics

In addition to the Boolean semantics, quantitative semantics of STL quantifies the *robustness degree* of satisfaction by a particular trace

$$
\begin{aligned}
\rho(\top, x, t) &= +\infty \\
\rho(x_i \geq c, x, t) &= x_i(t) - c \\
\rho(\neg\phi, x, t) &= -\rho(\phi, x, t) \\
\rho(\phi_1 \wedge \phi_2, x, t) &= \min\{\rho(\phi_1, x, t), \rho(\phi_2, x, t)\} \\
\rho(\phi_1 U_I \phi_2, x, t) &= \sup_{t_1 \in t+I} \min\{\rho(\phi_2, x, t_1), \inf_{t_2 \in [t, t_1)} \rho(\phi_1, x, t_2)\}
\end{aligned}
$$

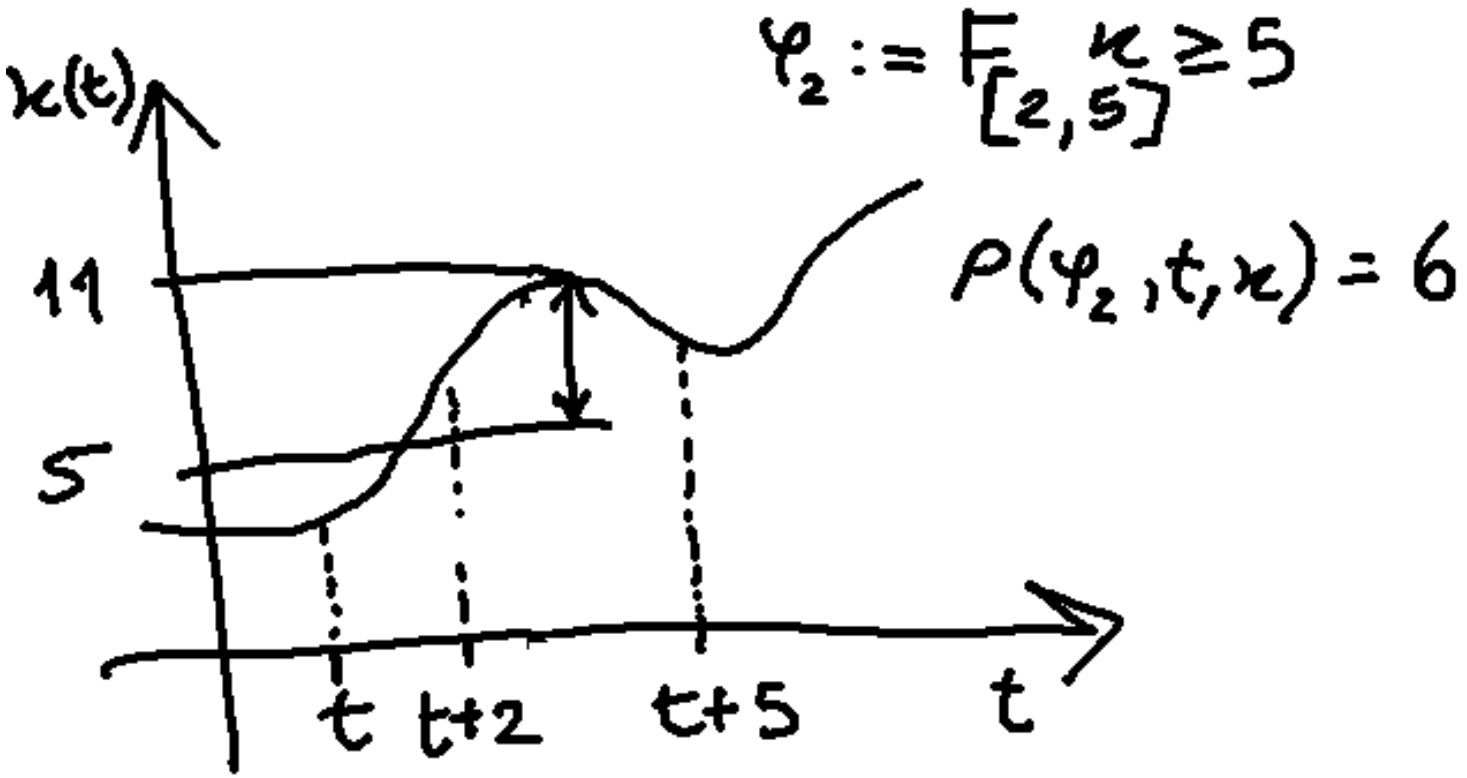
STL: Example 1

In addition to the Boolean semantics, quantitative semantics of STL quantify the *robustness degree* of satisfaction by a particular trace



STL: Example 2

In addition to the Boolean semantics, quantitative semantics of STL quantify the *robustness degree* of satisfaction by a particular trace

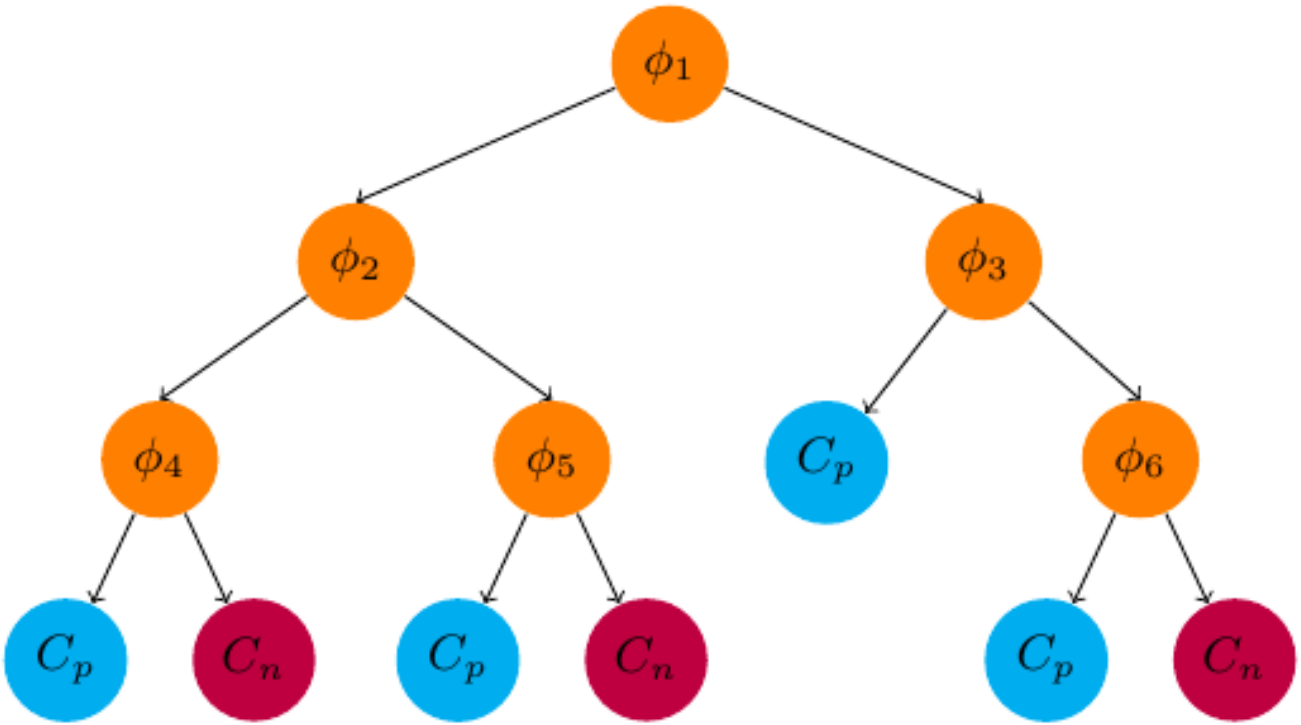


DTL4STL [Bombara et al. 2016]

For our purposes, we are interested in extracting constrasting STL specifications

- Framework for inference of timed temporal logic properties from data
- Produces a binary decision tree which can be translated in a STL formula and used for classification
- Each node of the tree is associated with a simple formula chosen from a set of *primitives*
- Optimality is assessed using *impurity* measures leveraging the *robustness degree* which capture how well a primitive splits the signals in the training data

DTL4STL - Decision Tree Classifier



For each node a formula that minimize the *impurity* measure is chosen from a set of *primitives*.

DTL4STL - Primitives

The optimization of the parameters in order to minimize the impurity measure is carried out through the Simulated Annealing algorithm. The overall complexity is $O(N(\log N))$.

Framework Operation

Framework Operation Settings

Execution Modes:

- warmup: mimic the continual arrival of the available traces from data pertaining to past system failures or generated by means of simulations
- online: incoming traces of the currently monitored system are considered

Execution Strategies:

- semi-supervised: domain experts specify an initial set of properties to be monitored against the execution of the system
- unsupervised: monitor initialized with just a single “the machinery is in operation” property

Application: Backblaze Hard Drive Dataset

Information regarding the health status of ST4000DM000 hard drive model in the Backblaze data center

Data recorded daily from 2015 to 2017

21 SMART parameters including both discrete and real values

Label which indicates a drive failure

Application: Experiment Setup

Initial phase in unsupervised learning warmup mode on a sample of training set execution traces which exhibited a failure

Two evaluation modes:

- online, in which the framework continues to learn properties from the execution traces of the test set
- offline, for SOTA comparison purposes

Counter-overfitting measures (trees):

- maximum height of 3
- minimum cross-validation accuracy score of 0.9
- maximum false alarm rate of 0.1 wrt pool of good training traces

Application: Of ine Results

$$\text{Precision} = \frac{TP}{TP + FP}; \quad \text{Recall} = \frac{TP}{TP + FN};$$

$$\text{FAR} = \frac{FP}{FP + TN}; \quad F1 = \frac{2 \quad \text{Precision} \quad \text{Recall}}{\text{Precision} + \text{Recall}} :$$

Application: RUL and Online Results

Application: Interpretability

Failure in which the hardware read error value stays below $1.6 \cdot 10^8$ for a certain amount of time, and then, at a later time, it exceeds $2.29 \cdot 10^8$)

The pattern represents a situation in which the errors encountered in reading data grow over time, till they exceed a warning threshold

Application: Interpretability (2)

Pattern witnessed during the warmup phase:

1. formula $f_1 = F_{[25;45]}\text{SMART}_{198} > 2{:}59$ is extracted
   critical sensor regarding sector read/write errors
2. formula f_1 triggers a failure prediction
3. as a consequence $f_2 = F_{[11;36]}\text{SMART}_{189} > 8{:}28$ is extracted
   non-critical sensor regarding unsafe y height conditions

The disk head is operating at an unsafe height, ultimately damaging a disk sector and consequently causing read and write errors (link between a non-critical and a critical sensor).

Future Work

Monotonicity of the monitoring pool property score?

Warmup learning on good traces

Redundancy in the monitoring pool

RUL estimation

Bibliography I

Giuseppe Bombara et al. (2016). “A decision tree approach to data classi cation using signal temporal logic”. In: Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, pp. 1–10.

Andrea Brunello, Dario Della Monica, and Angelo Montanari (2019). “Pairing Monitoring with Machine Learning for Smart System Veri cation and Predictive Maintenance”. In: Proceedings of the 1st Workshop on Arti cial Intelligence and Formal Veri cation, Logic, Automata, and Synthesis, vol. 2509. CEUR Workshop Proceedings, pp. 71–76.

Andrea Brunello, Dario Della Monica, et al. (2020). “Learning How to Monitor: Pairing Monitoring and Learning for Online System Veri cation”. In:

Bibliography II

Ian Cassar et al. (2017). “A Survey of Runtime Monitoring Instrumentation Techniques”. In: Proceedings of the 2nd International Workshop on Pre- Post-Deployment Veri cation Techniques, pp. 15–28.

Alexandre Donzé (2013). “On signal temporal logic”. In: International Conference on Runtime Veri cation. Springer, pp. 382–383.

Marcus Gerhold, Arnd Hartmanns, and Mariëlle Stoelinga (2019). “Model-Based Testing of Stochastically Timed Systems”. In: Innovations in Systems and Software Engineering 15.3-4, pp. 207–233. DOI: 10.1007/s11334-019-00349-z. URL: https://doi.org/10.1007/s11334-019-00349-z .

N. Jansen et al. (2018). “Machine Learning and Model Checking Join Forces”. In: Dagstuhl Reports 8.3, pp. 74–93.

Bibliography III

Martin Leucker and Christian Schallhart (2009). "A brief account of Runtime Verification". In: *Journal of Logical and Algebraic Methods in Programming* 78.5, pp. 293–303. ISSN: 1567-8326. DOI: http://dx.doi.org/10.1016/j.jlap.2008.08.004. URL: http://www.sciencedirect.com/science/article/pii/S1567832608000775.

Oded Maler and Dejan Nickovic (2004). "Monitoring temporal properties of continuous signals". In: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, pp. 152–166.

Amir Pnueli (1977). "The temporal logic of programs". In: *Proceedins of the 18th Annual Symposium on Foundations of Computer Science*. IEEE, pp. 46–57.

Bibliography IV

Property Pattern Mappings for LTL (n.d.).
https://matthewbdwyer.github.io/psp/. Kansas State University CIS Department, Laboratory for Specification, Analysis, and Transformation of Software (SAnToS Laboratory) – accessed online on 17 July 2020.