



University of Udine

---

# Learning from Failures

*Machine Learning-based Monitoring  
for Runtime System Verification*

Andrea Brunello - [andrea.brunello@uniud.it](mailto:andrea.brunello@uniud.it)  
Andrea Urgolo - [urgolo.andrea@spes.uniud.it](mailto:urgolo.andrea@spes.uniud.it)



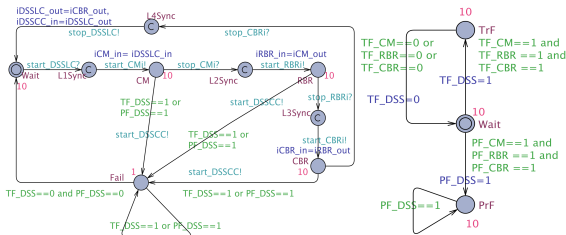
In several domains, systems generate continuous streams of data which may contain useful telemetry information

- 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

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!

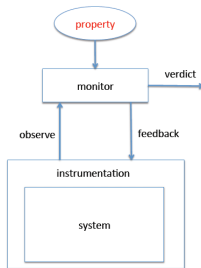


Framework that combines machine learning and monitoring to detect critical system behaviours in an on-line setting:

- 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 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).

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

$$\pi, s_i \models p \in P \Leftrightarrow V(p, s_i) = \text{true}$$

$$\pi, s_i \models \neg\alpha \iff \pi, s_i \not\models \alpha$$

$$\pi, s_i \models \alpha \wedge \beta \iff \pi, s_i \models \alpha \wedge \pi, s_i \models \beta$$

$$\pi, s_i \models X\alpha \iff \pi, s_{i+1} \models \alpha$$

$$\pi, s_i \models F\alpha \iff \exists j \geq i : \pi, s_j \models \alpha$$

$$\pi, s_i \models G\alpha \iff \forall j \geq i : \pi, s_j \models \alpha$$

$$\pi, s_i \models \alpha U \beta \iff \exists j \geq i : \pi, s_j \models \beta \text{ e } \forall k \text{ s.t. } i \leq k < j : \pi, s_k \models \alpha$$

Positively monitorable properties:

- every system satisfying it features a finite trace witnessing the satisfaction
- *it is possible to reach a success state*:  $F \text{ success}$

Negatively monitorable properties:

- every system violating it features a finite trace witnessing the violation
- *it is not possible to reach a success state*:  $G \neg \text{success}$

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*:  
 $F \text{ success} \wedge F(G \neg \text{success})$



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

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

$$(\mathbf{x}, t) \models f(\mathbf{x}) \sim c \iff f(\mathbf{x}(t)) \sim c \text{ is true}$$

$$(\mathbf{x}, t) \models \neg\varphi \iff (\mathbf{x}, t) \not\models \varphi$$

$$(\mathbf{x}, t) \models \varphi_1 \wedge \varphi_2 \iff (\mathbf{x}, t) \models \varphi_1 \wedge (\mathbf{x}, t) \models \varphi_2$$

$$(\mathbf{x}, t) \models \varphi_1 \mathbf{U}_I \varphi_2 \iff \exists t_1 \in t \oplus I : (\mathbf{x}, t_1) \models \varphi_2 \wedge \\ \forall t_2 \in [t, t_1) : (\mathbf{x}, t_2) \models \varphi_1$$

where  $\sim \in \{\leq, \geq, =\}$  and  $I := (a, b) \mid (a, b] \mid [a, b) \mid [a, b]$  with  $a, b \in \mathbb{R}^{\geq 0}$  and  $a \leq b$



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

$$\rho(\top, x, t) = +\infty$$

$$\rho(x_i \geq c, x, t) = x_i(t) - c$$

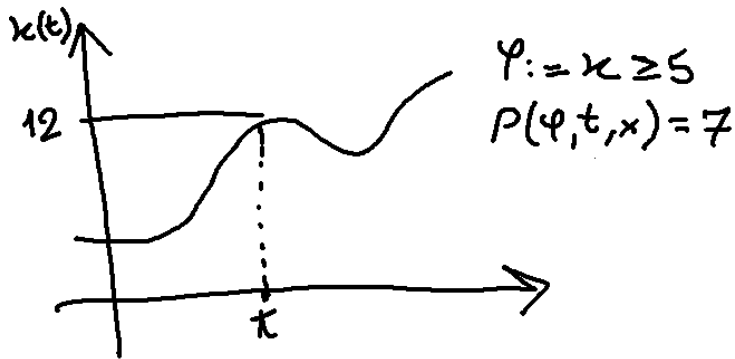
$$\rho(\neg\phi, x, t) = -\rho(\phi, x, t)$$

$$\rho(\phi_1 \wedge \phi_2, x, t) = \min\{\rho(\phi_1, x, t), \rho(\phi_2, x, t)\}$$

$$\rho(\phi_1 U_I \phi_2, x, t) = \sup_{t_1 \in t+I} \min\{\rho(\phi_2, x, t_1), \inf_{t_2 \in [t, t_1)} \rho(\phi_1, x, t_2)\}$$

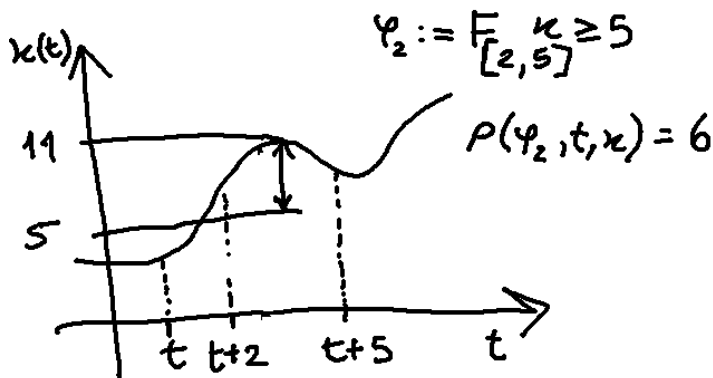
# STL: Example 1

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



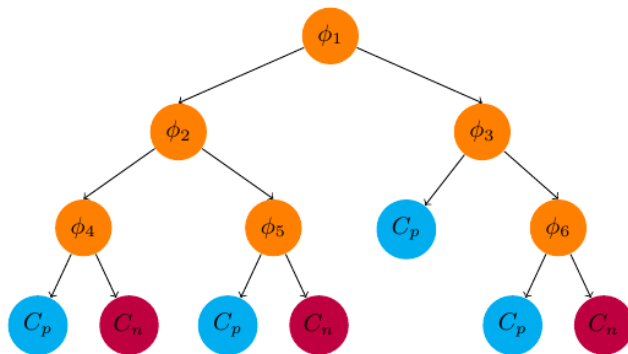
## STL: Example 2

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



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



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

$$\mathcal{P}_1 = \left\{ \mathbf{F}_{[\tau_1, \tau_2)}(x_i \sim \mu) \text{ or } \mathbf{G}_{[\tau_1, \tau_2)}(x_i \sim \mu) \right. \\ \left. \mid i \in \{1, \dots, n\}, \sim \in \{\leq, >\} \right\}$$

*The parameters of  $\mathcal{P}_1$  are  $(\mu, \tau_1, \tau_2)$  and the space of parameters is  $\Theta_1 = \mathbb{R} \times \{(a, b) \mid a < b, a, b \in \mathbb{R}_{\geq 0}\}$ .*

$$\mathcal{P}_2 = \left\{ \mathbf{G}_{[\tau_1, \tau_2)}\mathbf{F}_{[0, \tau_3)}(x_i \sim \mu) \text{ or } \mathbf{F}_{[\tau_1, \tau_2)}\mathbf{G}_{[0, \tau_3)}(x_i \sim \mu) \right. \\ \left. \mid i \in \{1, \dots, n\}, \sim \in \{\leq, >\} \right\}$$

*The parameters of  $\mathcal{P}_2$  are  $(\mu, \tau_1, \tau_2, \tau_3)$  and the space of parameters is  $\Theta_2 = \mathbb{R} \times \{(a, b) \mid a < b, a, b \in \mathbb{R}_{\geq 0}\} \times \mathbb{R}_{\geq 0}$ .*

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

■ **Algorithm 1** Framework execution

---

**Input:** initial pool of properties  $\mathcal{P}$ , incoming system trace  $t$

```
1: while True do
2:   if classifier  $m$  in  $\mathcal{P}$  predicts a failure in execution trace  $t$  then
3:     NOTIFYVIOLATION()
4:      $t_{normal}, t_{failure} \leftarrow \text{GETSUBTRACES}(t)$ 
5:      $train\_data \leftarrow \text{AUGMENT}(t_{normal}, t_{failure})$ 
6:      $f \leftarrow \text{EXTRACTDISCRFORMULAS}(train\_data)$ 
7:      $m \leftarrow \text{BUILDCLASSIFIER}(train\_data, f)$ 
8:     if  $m$  is not overfitted then
9:       ADD( $\mathcal{P}, m$ )
10:    end if
11:  end if
12: end while
```

---





## Execution Modes:

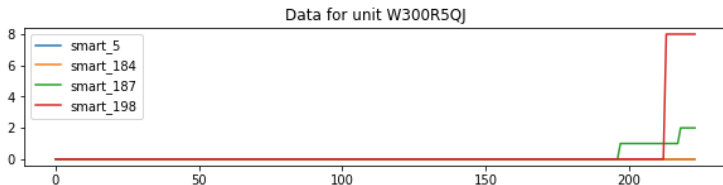
- *warmup*: mimic the continual arrival of the available traces from data pertaining to past system failures or generated by means of simulations
- *online*: incoming traces of the currently monitored system are considered

## 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

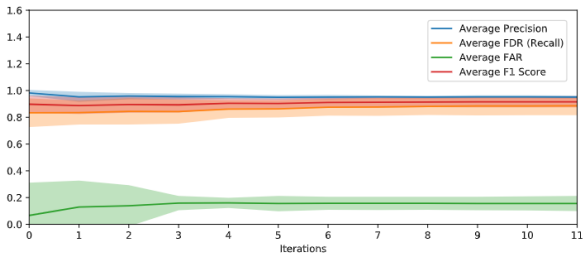
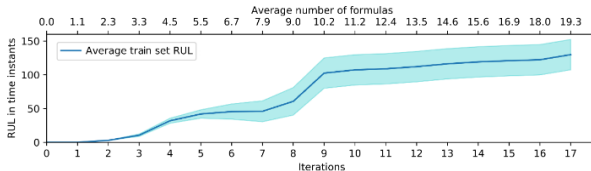
- Initial phase in *unsupervised learning warmup* mode on a sample of training set execution traces which exhibited a failure
- Two evaluation modes:
  - *online*, in which the framework continues to learn properties from the execution traces of the test set
  - *offline*, for SOTA comparison purposes
- 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

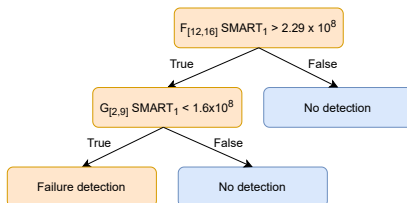
	SMART LTL (S1)	SMART STL (S1)	SMART XGBoost (S1) [11]	SMART NN (S1) [11]	SMART LTL (S2)	SMART STL (S2)	SMART LSTM (S2) [30]
<b>Precision</b>	0.29	0.36	0.40	0.50	0.54	0.97	0.91
<b>Recall</b>	0.99	0.83	0.60	0.53	0.96	0.83	0.94
<b>FAR</b>	0.57	0.42	0.01	0.01	0.15	0.08	0.05
<b>F<sub>1</sub> score</b>	0.44	0.51	0.48	0.52	0.69	0.89	0.93

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

$$\text{FAR} = \frac{FP}{FP + TN}, \quad \text{F1} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}.$$

# Application: RUL and Online Results





- Failure in which the *hardware read errors* value stays below  $1.6 \times 10^8$  for a certain amount of time, and then, at a later time, it exceeds  $2.29 \times 10^8$ )
- The pattern represents a situation in which the errors encountered in reading data grow over time, till they exceed a warning threshold

Pattern witnessed during the *warmup* phase:

- ① formula  $f_1 = F_{[25,45]} SMART_{198} > 2.59$  is extracted
  - critical sensor regarding *sector read/write errors*
- ② formula  $f_1$  triggers a failure prediction
- ③ as a consequence,  $f_2 = F_{[11,36]} SMART_{189} > 8.28$  is extracted
  - non-critical sensor regarding *unsafe fly 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).



- Monotonicity of the monitoring pool  $\rightsquigarrow$  property score?
- Warmup learning on good traces
- Redundancy in the monitoring pool
- RUL estimation





- Giuseppe Bombara et al. (2016). “A decision tree approach to data classification 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 Verification and Predictive Maintenance”. In: *Proceedings of the 1st Workshop on Artificial Intelligence and Formal Verification, 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 Verification”. In:

- Ian Cassar et al. (2017). “A Survey of Runtime Monitoring Instrumentation Techniques”. In: *Proceedings of the 2nd International Workshop on Pre- Post-Deployment Verification Techniques*, pp. 15–28.
- Alexandre Donzé (2013). “On signal temporal logic”. In: *International Conference on Runtime Verification*. 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.



- 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: *Proceedings 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.