Learning what to monitor: pairing monitoring and learning

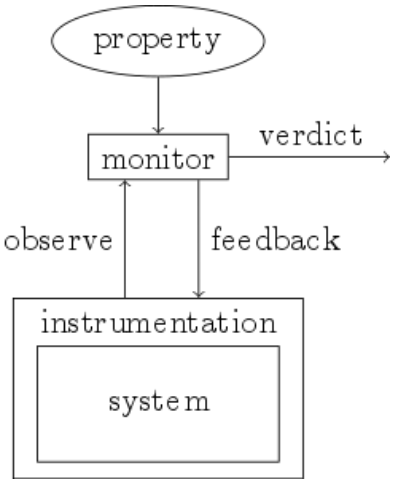
Angelo Montanari
(with A. Brunello, D. Della Monica, L. Geatti, and N. Saccomanno)

January 16, 2024

Monitoring is a runtime verification technique 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.

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).

A simple architecture



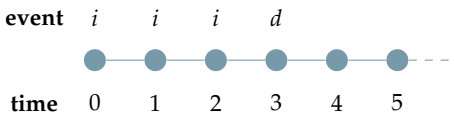
Monitoring typically consists of the following steps:

1. bad/good behaviors to be checked are specified by temporal logic formulas;
2. from each temporal formula, an equivalent *monitor* is built (typically, a deterministic finite state automaton);
3. the monitor is used for analysing the system in either online or offline mode.

Positively and negatively monitorable properties

Negatively monitorable

Property: *"The program never enters a deadlock"*

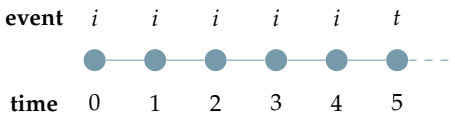


i = instruction
d = deadlock

Given a *safety* property, a prefix of a sequence suffices to establish whether it *does not* satisfy the property.

Positively monitorable

Property: *"The program terminates"*



i = instruction
t = termination

Given a *cosafety* property, a prefix of a sequence suffices to establish whether it *does* satisfy the property.

The cosafety fragment of LTL

We say that a temporal logic $\mathbb{L}$ is *cosafety* iff, for any $\phi \in \mathbb{L}$, $\mathcal{L}(\phi)$ is *cosafety*.

cosafetyLTL

Definition

$\phi := p \mid \neg p \mid \phi \vee \phi \mid \phi \wedge \phi \mid X\phi \mid F\phi \mid \phi U \phi$

Example:

pUq

F(pLTL)

Definition

$\phi := F(\alpha)$, where α is a pure-past LTL formula.

Example:

$F(q \wedge \tilde{Y}Hp)$

F(pLTL) is the canonical form of cosafetyLTL.

The safety fragment of LTL

We say that a temporal logic $\mathbb{L}$ is *safety* iff, for any $\phi \in \mathbb{L}$, $\mathcal{L}(\phi)$ is *safety*.

safetyLTL

Definition

$\phi := p \mid \neg p \mid \phi \vee \phi \mid \phi \wedge \phi \mid X\phi \mid G\phi \mid \phi R \phi$

Example:

$G(r- > XXg)$

G(pLTL)

Definition

$\phi := G(\alpha)$, where α is a pure-past LTL formula.

Example:

$G(\tilde{Y}\tilde{Y}r- > g)$

G(pLTL) is the canonical form of safetyLTL.

Monitorability behind safety and cosafety fragments

The class of monitorable LTL properties is larger than the union of safety and cosafety properties.

$$((p \vee q)Ur) \vee Gp$$

On the one hand, observe that:

- $ppp\ldots$ satisfies the formula (but none of its prefixes is good - not a cosafety property)
- $qqq\ldots$ does not satisfy the formula (but none of its prefixes is bad - not a safety property)

On the other hand:

- $\ldots r$ is a good prefix for the formula, provided that one of p or q holds in positions denoted by $\ldots$
- $\ldots \{\neg p, \neg q, \neg r\}$ is a bad prefix for the formula

Key point. Any finite prefix that is neither good nor bad can be extended to a good or a bad prefix: any letter containing r makes the prefix good, while a continuation with the letter $\{\neg p, \neg q, \neg r\}$ makes the prefix bad.

Properties which are not monitorable

There are properties which are neither positively nor negatively monitorable.

This is, for instance, the case with the property (*reactivity* property):

Every request is sooner or later granted

- If a request has not been yet granted, you cannot exclude that it will be granted in the future.
- If up to now all requests have been granted, you cannot exclude that a future one will not.

Limitation of monitoring

Monitoring suffers from some significant limitations.

Among them we would like to mention the following ones:

- modern systems have such a level of complexity that it is impossible for a system engineer to specify in advance all properties to be monitored;
- even minor changes to the system can introduce unforeseen bugs.

Learning what to monitor

How to solve this problem? By *pairing monitoring and learning*.

Main idea

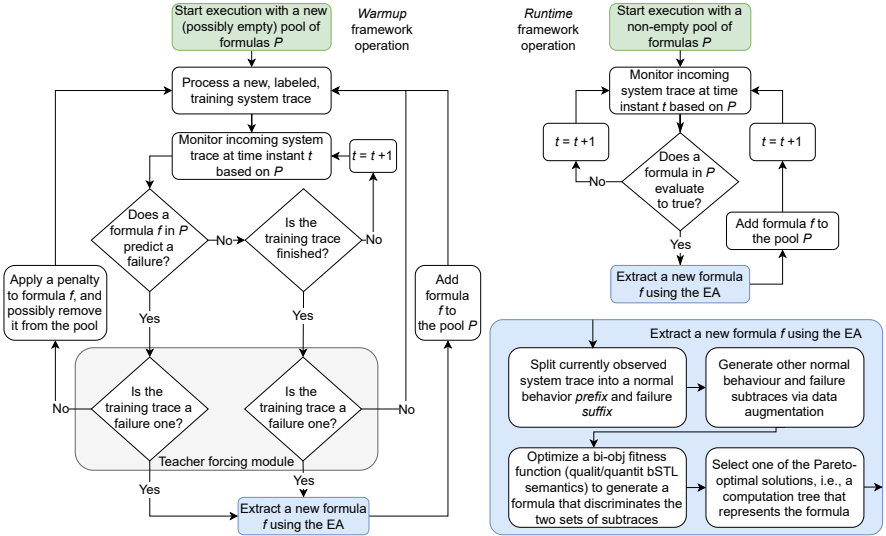
Machine learning is used to learn in an online, iterative fashion, new formulas to monitor by analysing trace prefixes that lead to failure events.

Pairing monitoring with machine learning

- The system engineering specifies only a set of initial properties.
- *Warmup (offline) Phase*: In a fully automatic fashion, the framework analyzes the traces of the system that lead to a failure, and derive new relevant properties, with the objective of *anticipate* their identification.
- *Online Phase*: the framework monitors the system in real time. If a failure occurs, it derives new relevant formulas and it iteratively refines the pool of formulas to be monitored.

A proof of concept with Signal Temporal Logic and Genetic Programming for early failure detection and predictive maintenance.

The framework



Distinguishing features of the framework

Distinguishing features of the framework:

- *interpretability*: the machine learning methods manipulate and produce only formulae, that can be easily inspected by a system engineer;
- formal guarantees on *monitorability*: every formula produced during the learning phase is guaranteed to be monitorable (this is done syntactically, through the grammar used for the generation of the computation tree of each formula – from bounded future STL to safety and cosafety fragments of STL);
- *generality*: different monitoring and machine learning backends.

Limitations of "pure" machine learning approaches

The machine learning techniques that are most used in predictive maintenance and early failure detection are:

- Random Forests
- Artificial Neural Networks
- Support Vector Machines

The *lack of interpretability* is a problem common to all of the above techniques, in the sense that they fail to provide an explanation of their prediction.

Explanation of the output of a predictive maintenance algorithm is important not only for humans trying to understand the error but also for implementing the correct actions for preventing the failure of the asset under consideration.

Extensions to the framework

The framework can be extended along the following directions:

- identification of *anomalies* and *drops of performance*
- exploitation of *unsupervised and self-supervised learning* techniques
- modularity

Anomaly detection and self-supervised learning backend

- The framework currently works in a **supervised fashion**
  - traces, labeled as failure or good behaviour ones, guide the first (*warmup*) stage of formula extraction, following a teacher-forcing like approach
  - justified as failures are terminating events (always detectable)
- The new version shall be **self-supervised** to deal with anomalies:
  - characterizing *a priori* anomalies in modern complex systems is impractical
  - systems evolve continuously over time
  - supervised assumption, i.e., existence of complete and exhaustive dataset of labelled anomalies is unrealistic and unfeasible
- Possible solution: use deep learning based SOTA approaches capable to perform self-supervised anomaly detection as a source of supervision

Assumption-based runtime verification

Assumption-Based Runtime Verification (ABRV) has recently been introduced as a variant of monitoring to deal with systems that are only *partially observable*

- 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
- ABRV can reach conclusive verdicts with shorter trace prefixes

Modularity

The new framework shall be *modular* in at least the following dimensions:

- the specification language
  - different temporal logics including LTL, STL, and ITL
  - *qualitative* semantics (for tasks like failure detection)
  - *quantitative* semantics (for tasks like anomaly detection), where appropriate
- the backend implementing the monitoring algorithm
- the backend for the learning of new properties
  - move across learning paradigms and tasks
  - 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; alternatively, formulas can be represented as graphs, enabling the usage of Graph Neural Networks

Some references

- Bauer, A., Leucker, M., & Schallhart, C. (2011). Runtime verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology.
- Cimatti, A., Tian, C., & Tonetta, S. (2022). Assumption-based Runtime Verification. Formal Methods in System Design.
- Geatti L. & Montanari, A. (2023). The Safety Fragment of Temporal Logics of Infinite Sequences . ESSLLI Summer School
- Brunello, A., Della Monica, D., Montanari, A., Saccomanno, N., & Urgolo, A. (2023). Monitors that Learn from Failures: Pairing STL and Genetic Programming. IEEE Access.
- Geatti L., Montanari, A., & Saccomanno, N. (2023). Towards Machine Learning Enhanced LTL Monitoring. OVERLAY workshop