An Operational Guide to Monitorability

Luca Aceto[1,2](✉), Antonis Achilleos[2](✉), Adrian Francalanza[3](✉), Anna Ingólfsdóttir[2](✉), and Karoliina Lehtinen[4](✉)

[1] Gran Sasso Science Institute, L'Aquila, Italy
[2] Reykjavik University, Reykjavik, Iceland
{luca,antonios,anna}@ru.is
[3] University of Malta, Msida, Malta
adrian.francalanza@um.edu.mt
[4] University of Liverpool, Liverpool, UK
k.lehtinen@liverpool.ac.uk

Abstract. Monitorability underpins the technique of Runtime Verification because it delineates what properties can be verified at runtime. Although many monitorability definitions exist, few are defined *explicitly* in terms of the operational guarantees provided by monitors, *i.e.,* the computational entities carrying out the verification. We view monitorability as a spectrum, where the fewer guarantees that are required of monitors, the more properties become monitorable. Accordingly, we present a monitorability hierarchy based on this trade-off. For regular specifications, we give syntactic characterisations in Hennessy–Milner logic with recursion for its levels. Finally, we map existing monitorability definitions into our hierarchy. Hence our work gives a unified framework that makes the operational assumptions and guarantees of each definition explicit. This provides a rigorous foundation that can inform design choices and correctness claims for runtime verification tools.

1 Introduction

Runtime Verification (RV) [12] is a lightweight verification technique that checks for a specification by analysing the current execution exhibited by the system under scrutiny. Despite its merits, the technique is limited in certain respects: any sufficiently expressive specification language contains properties that cannot be monitored at runtime [2,3,19,24,30,39,41]. For instance, the *satisfaction* of a safety property ("bad things never happen") cannot, in general, be determined by observing the (finite) behaviour of a program up to the current execution point; its *violation*, however, can. *Monitorability* [12,41] concerns itself

This research was supported by the Icelandic Research Fund projects "TheoFoMon: Theoretical Foundations for Monitorability" (№:163406-051) and "Epistemic Logic for Distributed Runtime Monitoring" (№:184940-051), by the BMBF project "Aramis II" (№:01IS160253), the EPSRC project "Solving parity games in theory and practice" (№:EP/P020909/1), and project BehAPI, funded by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement №:778233.

P. C. Ölveczky and G. Salaün (Eds.): SEFM 2019, LNCS 11724, pp. 433–453, 2019.
https://doi.org/10.1007/978-3-030-30446-1_23

with the delineation between properties that are monitorable and those that are not. Besides its importance from a foundational perspective, monitorability is paramount for a slew of RV tools, such as those described in [9,17,23,40,42], that synthesise monitors from specifications expressed in a variety of logics. These monitors are executed with the system under scrutiny to produce verdicts concerning the satisfaction or violation of the specifications from which they were synthesised.

Monitorability is crucial for a principled approach because it disciplines the construction of RV tools. It should espouse the separation of concerns between the specification of a correctness property on the one hand, and the method used to verify it on the other [30]. It defines, either explicitly or implicitly, a notion of *monitor correctness* [27,28,31,38], which then guides the automated synthesis of monitors from specifications. It also delimits the monitorable fragment of the specification logic on which the synthesis is defined: monitors need not be synthesised for non-monitorable specifications. In some settings, a syntactic characterisation of monitorable properties can be identified [1,3,30], and used as a *core calculus* for studying optimisations of the synthesis algorithm. More broadly, monitorability boundaries may guide the design of *hybrid* verification strategies, which combine RV with other verification techniques (see the work in [2] for an example of this approach).

In spite of its importance, there is *no* generally accepted notion of monitorability to date. The literature contains a number of definitions, such as the ones proposed in [3,14,25,30,32,41]. These differ in aspects such as the adopted specification formalism, *e.g.,* LTL, Street automata, recHML *etc.,*, the operational model, *e.g.,* testers, automata, process calculi *etc.,*, and the semantic domain, *e.g.,* infinite traces, finite and infinite (finfinite) traces or labelled transition systems. Even after these differences are normalised, many of these definitions are *not* in agreement: there are properties that are monitorable according to some definitions but *not* monitorable according to others. More alarmingly, as we will show, frequently cited definitions of monitorability contain serious errors.

This discrepancy between definitions raises the question of *which one* to adopt when designing and implementing an RV tool, and *what effect* this choice has on the behaviour of the resulting tool. A difficulty in informing this choice is that few of those definitions make explicit the relationship between the operational model, *i.e.,* the behaviour of a monitor, and the monitored properties. In other words, it is not clear what the guarantees provided by the various monitors mentioned in the literature are, and how they differ from each other.

Example 1. Consider the runtime verification of a system exhibiting (only) three events over finfinite traces: failure (f), success (s) and recovery (r). One property we may require is that *"failure never occurs and eventually success is reached"*, otherwise expressed in LTL fashion as $(\mathsf{G}\,\neg\mathsf{f}) \wedge (\mathsf{F}\,\mathsf{s})$. According to the definition of monitorability attributed to Pnueli and Zaks [41] (discussed in Sect. 7), this property is monitorable. However, it is not monitorable according to others, including Schneider [44], Viswanathan and Kim [45], and Aceto *et al.* [3], whose definition of monitorability coincides with some subset of *safety properties.* ■

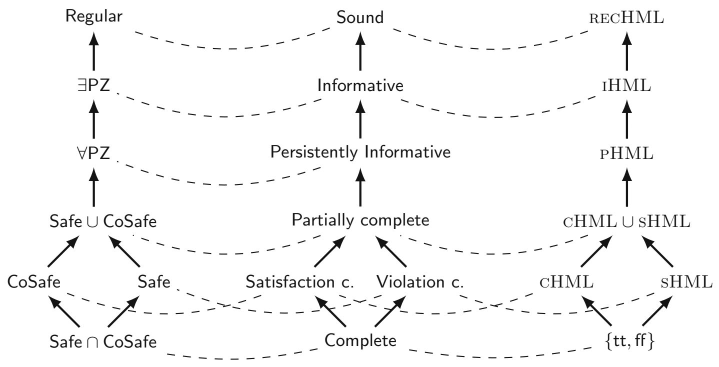

Fig. 1. The monitorability hierarchy of regular properties

Contributions. To our mind, this state of the art is unsatisfactory for tool construction. More concretely, an RV tool broadly relies on the following ingredients:

1. the *input* of the tool in terms of the formalism used to describe the specification properties;
2. the executable description of monitors that are the tool's *output* and
3. the *mapping* between the inputs and outputs, *i.e.,* the synthesis function of monitors from specifications.

Any account on monitorability should, in our view, shed light on those three aspects, particularly on what it means for the synthesis function and the monitors it produces to be *correct*. This involves establishing the relationship between *the truth value* of a specification, given by a two-valued semantics, and *what the runtime analysis tells us about it*, given by the operational behaviour exhibited by the monitor; ideally, the specification and operational descriptions should also be described independently of one another, in order to ensure the aforementioned separation of concerns.[1] In addition, any account on monitorability should also be flexible enough to incorporate a variety of relationships between specification properties and the expected behaviour of monitors. This is essential for it be of use to the tool implementors, acting as a principled foundation to guide their design decisions.

For these reasons, we take the view that monitorability comes on a *spectrum.* There is a trade-off between the guarantees provided by monitors and the properties that can be monitored with those guarantees. We argue that considering different requirements gives rise to a *hierarchy of monitorability*—depicted in Fig. 1 (middle)—which classifies properties according to what types of guarantees RV can give for them. At one extreme, anything can be monitored if the

[1] In RV, it is commonplace to see the expected monitor behaviour described via an intermediary n-valued logic semantics [13,14,32] (*e.g.,* mapping finite traces into the three verdicts called accepting, rejecting and inconclusive). Although convenient in certain cases, the approach goes against our tenet for the separation of concerns.

only requirement is for monitors to be *sound i.e.,* they should not contradict the monitored specification. However, monitors that are *just* sound give no guarantees of ever giving a verdict. More usefully, *informatively* monitorable properties enjoy monitors that reach a verdict for *some* finite execution; arguably, this is the minimum requirement for making monitoring potentially worthwhile. More stringent requirements can demand this capability to be *invariant* over monitor executions, *i.e.,* a monitor never reaches a state where it cannot provide a verdict; then we speak of persistently informative monitors. Adding completeness requirements of different strengths, such as the requirement that a monitor should be able to identify all failures and/or satisfactions, yields stronger definitions of monitorability: partial, satisfaction or violation complete, and complete.

In order not to favour a specific operational model, the hierarchy in Fig. 1 (middle) is cast in terms of abstract behavioural requirements for monitors. We then provide an instantiation that concretises those requirements into an *operational* hierarchy, establishing operational counterparts for each type of monitorability over regular properties. To this end, we use the operational framework developed in [3], that uses finite-state monitors and in which partial and complete monitorability were already defined. We show this framework to be, in a suitable technical sense, maximally general (Theorem 2) for regular properties. This shows that our work is equally applicable to other operational models for monitoring regular properties.

In order for a tool to synthesise monitors from specifications, it is useful to have *syntactic characterisations* of the properties that are monitorable with the required guarantees: synthesis can then directly operate on the syntactic fragment. We offer monitorability characterisations as fragments of recHML [6,37] (a variant of the modal μ-calculus [34]) interpreted over finfinite traces—see Fig. 1 (right). The logic is expressive enough to capture all regular properties—the focus of nearly all existing definitions of monitorability—and subsumes more user-friendly but less expressive specification logics such as LTL. Partial and complete monitorability already enjoy monitor synthesis functions and neat syntactic characterisations in recHML [3]; related synthesis functions based on syntactic characterisations for a branching-time setting [29,30] have already been implemented in a tool [8,9]. Here, we provide the missing syntactic characterisation for informative monitorability, and for a fragment of persistently informative monitorability.

Finally, we show that the proposed hierarchy accounts for existing notions of monitorability. See Fig. 1 (left). Safety, co-safety and their union correspond to partial monitorability and its two components, satisfaction- and violation-monitorability; Pnueli and Zaks's definition of monitorability can be interpreted in two ways, of which one ($\exists$pz) maps to informative monitorability, and the other ($\forall$pz) to persistently informative monitorability. We also show that the definitions of monitorability proposed by Falcone *et al.* [25], contrary to their claim, do *not* coincide with safety and co-safety properties. To summarise, our principal contributions are:

1. A unified operational perspective on existing notions of monitorability, clarifying what operational guarantees each provides, see Theorems 1, 6 and 7;
2. An extension to the syntactic characterisations of monitorable classes from [3], mapping all but one of these classes to fragments in RECHML, which can be viewed as a target byte-code for higher-level logics, see Theorems 4 and 5.

2 Preliminaries

Traces. We assume a *finite* set of actions, $a, b, \ldots \in \text{Act}$. The metavariables $\mathtt{t}, u \in \text{Act}^\omega$ range over *infinite* sequences of actions. *Finite traces*, denoted as $s, r \in \text{Act}^*$, represent *finite* prefixes of system runs. Collectively, finite and infinite traces $\text{Act}^\infty = \text{Act}^\omega \cup \text{Act}^*$ are called *finfinite* traces. We use $f, g \in \text{Act}^\infty$ to range over finfinite traces and $F \subseteq \text{Act}^\infty$ to range over sets of finfinite traces. A (finfinite) trace with action a at its head is denoted as af. Similarly, a (finfinite) trace with a prefix s and continuation f is denoted as sf. We write $s \preceq f$ to denote that the finite trace s is a prefix of f, *i.e.,* $\exists g$ such that $f = sg$.

Properties. A *property* over finfinite (*resp.*, infinite) traces, denoted by the variable P, is a subset of Act^∞ (*resp.*, of Act^ω). In general, a *property* refers to a finfinite property, unless stated otherwise. A finite trace s *positively determines* a property $P \subseteq \text{Act}^\infty$ when $sf \in P$ for *every* continuation $f \in \text{Act}^\infty$; analogously, s *negatively determines* P when $sf \notin P$ for every $f \in \text{Act}^\infty$. The same terms apply similarly when $P \subseteq \text{Act}^\omega$. We call a property *regular* if it is the union of a regular property $P_{\text{fin}} \subseteq \text{Act}^*$ and an ω-regular property $P_{\text{inf}} \subseteq \text{Act}^\omega$.

3 A Monitor-Oriented Hierarchy

From a tool-construction perspective, it is important to give concrete, implementable definitions of monitors; we do so in Sect. 4. To understand the guarantees that these monitors will provide, we first discuss the general notion of monitor and monitoring system. Already in this general setting, we are able to identify the various requirements that give rise to the hierarchy of monitorability, depicted in the middle part of Fig. 1. Section 4 will then provide operational semantics to this hierarchy, in the setting of regular properties.

We consider a monitor to be an entity that analyses finite traces and (at the very least) identifies a set of finfinite traces that it *accepts* and a set of finfinite traces that it *rejects*. We consider two postulates. Firstly, an acceptance or rejection verdict has to be based on a finite prefix of a trace, Definition 1. 1: verdicts are thus given for *incomplete* traces. Secondly, verdicts must be *irrevocable*, Definition 1. 2. These postulates make explicit two features shared by most monitorability definitions in the literature.

Definition 1. *A* monitoring system *is a triple* $(M, \boldsymbol{acc}, \boldsymbol{rej})$*, where* M *is a nonempty set of monitors,* $\boldsymbol{acc}, \boldsymbol{rej} \subseteq M \times \text{Act}^\infty$*, and for every* $m \in M$ *and* $f \in \text{Act}^\infty$*:*

1. $\big(\boldsymbol{acc}(m,f)$ *implies* $\exists s \cdot \big(s \preceq f$ *and* $\boldsymbol{acc}(m,s)\big)\big)$ *and* $\big(\boldsymbol{rej}(m,f)$ *implies* $\exists s \cdot \big(s \preceq f$ *and* $\boldsymbol{rej}(m,s)\big)\big)$*;*
2. $\big(\boldsymbol{acc}(m,s)$ *implies* $\forall f \cdot \boldsymbol{acc}(m,sf)\big)$ *and* $\big(\boldsymbol{rej}(m,s)$ *implies* $\forall f \cdot \boldsymbol{rej}(m,sf)\big)$*.* ∎

Remark 1. Finite automata do not satisfy the requirements of Definition 1 since their judgement can be revoked. Standard Büchi automata are not good candidates either, since they need to read the entire infinite trace to accept or reject. ∎

We define a notion of maximal monitoring system for a collection of properties; for each property P in that set, such a system must contain a monitor that reaches a verdict for all traces that have some prefix that determines P.

Definition 2. *A monitoring system* $(M, \boldsymbol{acc}, \boldsymbol{rej})$ *is* maximal *for a collection of properties* $C \subseteq 2^{\textsc{Act}^\infty}$ *if for every* $P \in C$ *there is a monitor* $m_P \in M$ *such that*

(i) $\boldsymbol{acc}(m_P, f)$ *iff trace* f *has a prefix that positively determines* P*;*
(ii) $\boldsymbol{rej}(m_P, f)$ *iff trace* f *has a prefix that negatively determines* P*.* ∎

In Sect. 4, we present an instance of such a maximal monitoring system for regular properties. This shows that, for regular properties at least, the maximality of a monitoring system is a reasonable requirement. Unless otherwise stated, we assume a fixed maximal monitoring system $(M, \mathbf{acc}, \mathbf{rej})$ throughout the rest of the paper. For $m \in M$ to monitor for a property P, it needs to satisfy some requirements. The most important such requirement is *soundness.*

Definition 3 (Soundness). *Monitor* m *is* sound *for property* P *if for all* f*:*

– $\boldsymbol{acc}(m,f)$ *implies* $f \in P$*,* and
– $\boldsymbol{rej}(m,f)$ *implies* $f \notin P$*.* ∎

Lemma 1. *If* m *is sound for* P *and* $\boldsymbol{acc}(m,s)$ *(resp.,* $\boldsymbol{rej}(m,s)$*), then* s *positively (resp., negatively) determines* P*.*

Lemma 2. *For every* $P \subseteq \textsc{Act}^\infty$*:* (i) m_P *is sound for* P*; and* (ii) *if* m *is a sound monitor for* P *and* $\boldsymbol{acc}(m,f)$ *(resp.,* $\boldsymbol{rej}(m,f)$*), then it is also the case that* $\boldsymbol{acc}(m_P,f)$ *(resp.,* $\boldsymbol{rej}(m_P,f)$*).*

The dual requirement to soundness, *i.e., completeness*, entails that the monitor detects *all* violating and satisfying traces. Unfortunately, this is only possible for trivial properties in the finfinite[2] domain—see Proposition 1. Instead, monitors may be required to accept all satisfying traces, or reject all violating traces.

Definition 4 (Completeness). *Monitor* m *is* satisfaction-complete *for* P *if* $f \in P$ *implies* $\boldsymbol{acc}(m,f)$ *and* violation-complete *for* P *if* $f \notin P$ *implies* $\boldsymbol{rej}(m,f)$*. It is* complete *for* P *if it is* both *satisfaction- and violation-complete for* P *and* partially-complete *if it is* either *satisfaction-* or *violation-complete.* ∎

[2] In the infinite domain more properties *are* completely monitorable, see Sect. 8.

Proposition 1. *If m is sound and complete for P then $P = \textsc{Act}^{\infty}$ or $P = \emptyset$.*

Proof. If $\varepsilon \in P$, then $\mathbf{acc}(m, \varepsilon)$, so from Definition 1, $\forall f \in \textsc{Act}^{\infty}.\ \mathbf{acc}(m, f)$. Due to the soundness of m, $P = \textsc{Act}^{\infty}$. Similarly, $P = \emptyset$ when $\varepsilon \notin P$. □

We define monitorability in terms of the guarantees that the monitors are expected to give. Soundness is not negotiable. Given the consequences of requiring completeness, as evidenced by Proposition 1, we consider weaker forms of completeness. The weaker the completeness guarantee, the more properties can be monitored.

Definition 5 (Complete Monitorability). *Property P is completely monitorable when there exists a monitor that is sound and complete for P. It is* monitorable for satisfactions *(resp.,* violations*) when there exists a monitor m that is sound and satisfaction (*resp., *and violation) complete for P. It is* partially *(-complete) monitorable when it is monitorable for satisfactions* or *violations.*

A class of properties $C \subseteq 2^{\textsc{Act}^{\infty}}$ is satisfaction, violation, partially, or completely monitorable, when every *property $P \in C$ is, respectively, satisfaction, violation, partially or completely monitorable. We denote the class of all satisfaction, violation, partially, and completely monitorable properties by maximal monitoring systems as SCmp, VCmp, PCmp, and Cmp, respectively.* ■

Since even partial monitorability, the weakest form in Definition 5, renders a substantial number of properties unmonitorable [3], one may consider even weaker forms of completeness that only flag a *subset* of satisfying (or violating) traces. Sound denotes monitorability *without* completeness requirements. Arguably, however, the weakest guarantee for a sound monitor of a property P to be of use is the one that pledges to flag at least *one* trace. One may then further strengthen this requirement and demand that this guarantee is *invariant* throughout the analysis of a monitor: for every prefix observed the monitor is still able to flag at least once (possibly after observing more actions).

Definition 6 (Informative Monitorability[3]). *Monitor m is satisfaction- (*resp., *violation-)* informative *if $\exists f \cdot \boldsymbol{acc}(m, f)$ (*resp., *$\boldsymbol{rej}(m, f)$). It is satisfaction- (*resp., *violation-)* persistently informative *if $\forall s \exists f \cdot \boldsymbol{acc}(m, sf)$ (*resp., *$\boldsymbol{rej}(m, sf)$). We simply say that m is informative (*resp., *persistently informative) when we do not distinguish between satisfactions or violations.* ■

Definition 7 (Informative Monitorability). *We say that property P is informatively (*resp., *persistently informatively) monitorable if there is an informative (*resp., *a persistently informative) monitor that is sound for P. A class of properties $C \subseteq 2^{\textsc{Act}^{\infty}}$ is informatively (*resp., *persistently informatively) monitorable, when all its properties are informatively (*resp., *persistently informatively) monitorable. The class of all informatively (*resp., *persistently informatively) monitorable properties by maximal monitoring systems is denoted as ICmp*

[3] These are *not* related to the *informative prefixes* from [35] or to *persistence* from [43].

$$\begin{array}{lllllll} \varphi, \psi \in \textsc{rec}\text{HML} ::= \mathsf{tt} & \mid \mathsf{ff} & \mid \varphi \vee \psi & \mid \varphi \wedge \psi & \\ \quad \mid \langle a \rangle \varphi & \mid [a]\varphi & \mid \mathsf{min}\, X.\varphi & \mid \mathsf{max}\, X.\varphi & \mid X \end{array}$$

$$\begin{array}{rclrcl} [\![\mathsf{tt}, \sigma]\!] & \overset{\text{def}}{=} & \textsc{Act}^{\infty} & [\![\mathsf{ff}, \sigma]\!] & \overset{\text{def}}{=} & \emptyset \\ [\![\varphi_1 \vee \varphi_2, \sigma]\!] & \overset{\text{def}}{=} & [\![\varphi_1, \sigma]\!] \cup [\![\varphi_2, \sigma]\!] & [\![\varphi_1 \wedge \varphi_2, \sigma]\!] & \overset{\text{def}}{=} & [\![\varphi_1, \sigma]\!] \cap [\![\varphi_2, \sigma]\!] \\ [\![[a]\varphi, \sigma]\!] & \overset{\text{def}}{=} & \{ f \mid f = ag \text{ implies } g \in [\![\varphi, \sigma]\!] \} & [\![\langle a \rangle \varphi, \sigma]\!] & \overset{\text{def}}{=} & \{ af \mid f \in [\![\varphi, \sigma]\!] \} \\ [\![\mathsf{min} X.\varphi, \sigma]\!] & \overset{\text{def}}{=} & \bigcap \{ F \mid [\![\varphi, \sigma[X \mapsto F]]\!] \subseteq F \} & & & \\ [\![\mathsf{max} X.\varphi, \sigma]\!] & \overset{\text{def}}{=} & \bigcup \{ F \mid F \subseteq [\![\varphi, \sigma[X \mapsto F]]\!] \} & [\![X, \sigma]\!] & \overset{\text{def}}{=} & \sigma(X) \end{array}$$

Fig. 2. recHML syntax and (finfinite) linear-time semantics

(resp., PICmp). A property P is persistently informatively monitorable for satisfaction (resp., for violation) if there is a satisfaction- (resp., violation-) persistently informative monitor that is sound for P. We revisit this definition in Sect. 4. ■

Example 2. The property *"f never occurs and eventually s is reached"* (Example 1) is *not* partially monitorable but *is* persistently informatively monitorable.

The property requiring that *"r only appears a finite number of times"* is *not* informatively monitorable. For if it were, the respective sound informative monitor m should at least accept or reject one trace. If it accepts a trace f, by Definition 1, it must accept some prefix $s \preceq f$. Again, by Definition 1, all continuations, including $s\mathsf{r}^{\omega}$, must be accepted by m. This makes it unsound, which is a contradiction. Similarly, if m rejects some f, it must reject some finite $s \preceq f$ that necessarily contains a finite number of r actions, making it unsound. ■

Theorem 1 (Monitorability Hierarchy). *The monitorability classes given in Definitions 5 and 7 form the inclusion hierarchy depicted in Fig. 1.*

Proof. The hardest inclusion to show from Fig. 1 is PCmp = SCmp∪VCmp ⊆ PICmp. Pick a property $P \in$ VCmp. Let $s \in \textsc{Act}^*$. If $\exists f \cdot sf \notin P$ then by Definition 4 we have $\mathbf{rej}(m_P, sf)$. Otherwise, $\forall f \cdot sf \in P$, meaning that s positively determines P, and by Definition 2 we have $\mathbf{acc}(m_P, sf)$. By Definition 6, we deduce that m_P is persistently informative since $\forall s \exists f \cdot \mathbf{acc}(m_P, sf)$ or $\mathbf{rej}(m_P, sf)$. Thus, by Definition 7, it follows that $P \in$ PICmp. The case for $P \in$ SCmp is dual. □

4 An Instantiation for Regular Properties

We provide a concrete maximal monitoring system for regular properties. This monitoring system gives an operational interpretation to the levels of the monitorability hierarchy, and enables us to find syntactic characterisations for them in recHML [3,37]. Since this logic is a reformulation of the modal μ-calculus [34], it is expressive enough to describe all regular properties and to embed specification formalisms such as LTL, (ω-)regular expressions, Büchi automata, and Street automata, used in the state of the art on monitorability.

The Logic. The syntax of recHML is defined by the grammar in Fig. 2, which assumes a countable set of logical variables $X, Y \in \textsc{LVar}$. Apart from the standard constructs for truth, falsehood, conjunction and disjunction, the logic is equipped with existential ($\langle a\rangle\varphi$) and universal ($[a]\varphi$) modal operators, and *two* recursion operators expressing least and greatest fixpoints (*resp.,* $\mathsf{min}\,X.\varphi$ and $\mathsf{max}\,X.\varphi$). The semantics is given by the function $[\![-]\!]$ defined in Fig. 2. It maps a (possibly open) formula to a set of (finfinite) traces [3] by induction on the formula structure, using valuations that map logical variables to sets of traces, $\sigma : \textsc{LVar} \rightarrow \mathcal{P}(\textsc{Act}^{\infty})$, where $\sigma(X)$ is the set of traces assumed to satisfy X. An existential modality $\langle a\rangle\varphi$ denotes all traces with a prefix action a and a continuation that satisfies φ, whereas a universal modality $[a]\varphi$ denotes all traces that are either *not* prefixed by a or are of the form ag for some g that satisfies φ. The sets of traces satisfying the least and greatest fixpoint formulae, $\mathsf{min}\,X.\varphi$ and $\mathsf{max}\,X.\varphi$, are the least and the greatest fixpoints, respectively, of the function induced by the formula φ. For closed formulae, we use $[\![\varphi]\!]$ in lieu of $[\![\varphi, \sigma]\!]$ (for some σ). Formulae are generally assumed to be closed and guarded [36]. In the discussions we occasionally treat formulae, φ, as the properties they denote, $[\![\varphi]\!]$.

LTL [20] is the specification logic of choice for many RV approaches. As a consequence, it is also the logic used by a number of studies in monitorability (*e.g.,* see [13,14,32]). Our choice of logic, recHML, is not limiting in this regard.

Example 3. The characteristic LTL operators can be encoded in recHML as:

$$\mathsf{X}\,\varphi \overset{\text{def}}{=} \textstyle\bigvee_{a\in\textsc{Act}} \langle a\rangle\varphi \qquad \varphi\,\mathsf{U}\,\psi \overset{\text{def}}{=} \mathsf{min}\,Y.\big(\psi \vee (\,\varphi \wedge \mathsf{X}\;Y)\big) \qquad \mathsf{F}\,\varphi \overset{\text{def}}{=} \mathsf{tt}\,\mathsf{U}\,\varphi$$
$$\varphi\,\mathsf{R}\,\psi \overset{\text{def}}{=} \mathsf{max}\,Y.\big((\,\psi \wedge \varphi\,) \vee (\,\psi \wedge \mathsf{X}\;Y)\big) \qquad \mathsf{G}\,\varphi \overset{\text{def}}{=} \mathsf{ff}\,\mathsf{R}\,\varphi$$

In examples, atomic propositions a and $\neg a$ *resp.,* denote $\langle a\rangle\mathsf{tt}$ and $[a]\mathsf{ff}$. ■

recHML allows us to consider monitorable properties that may be misses by previous approaches. For instance, it is well known that logics such as the modal μ-calculus (and variants such as recHML) can describe properties that are *not* expressible in popular specification languages like LTL [46].

Example 4. Recall the system discussed in Example 1 where $\textsc{Act} = \{\mathsf{f}, \mathsf{s}, \mathsf{r}\}$. Consider the property requiring that *"success* (s) *occurs on every even position"*. Although this is *not* expressible in LTL [46], it can be expressed in recHML as:

$$\varphi_{\text{even}} = \mathsf{max}\,X.\big(\textstyle\bigvee_{a\in\{\mathsf{f},\mathsf{s},\mathsf{r}\}} \langle a\rangle\langle\mathsf{s}\rangle X\,\big)$$

The weaker property *"success* (s) *occurs on every even position until the execution ends"* still cannot be expressed in LTL, but can be expressed in recHML:

$$\varphi_{\text{evenW}} = \mathsf{max}\,X.\big(\textstyle\bigwedge_{a\in\{\mathsf{f},\mathsf{s},\mathsf{r}\}} [a]\,([\mathsf{s}]X \wedge [\mathsf{f}]\mathsf{ff} \wedge [\mathsf{r}]\mathsf{ff})\,\big) \qquad ■$$

$$\begin{array}{rcllllll} m,n \in \textsc{Mon} & ::= & v & \mid a.m & \mid m+n & \mid m\otimes n & \mid m\oplus n & \mid \mathsf{rec}\,x.m \quad \mid x \\ v,u \in \textsc{Verd} & ::= & \mathsf{end} & \mid \mathsf{no} & \mid \mathsf{yes} & & & \end{array}$$

$$\textsc{mAct}\frac{}{a.m \xrightarrow{a} m} \qquad \textsc{mVer}\frac{}{v \xrightarrow{a} v} \qquad \textsc{mRec}\frac{m[\mathsf{rec}\,x.m/x] \xrightarrow{a} n}{\mathsf{rec}\,x.m \xrightarrow{a} n}$$

$$\textsc{mSelL}\frac{m \xrightarrow{a} m'}{m+n \xrightarrow{a} m'} \qquad \textsc{mPar}\frac{m \xrightarrow{a} m' \quad n \xrightarrow{a} n'}{m\odot n \xrightarrow{a} m'\odot n'}$$

$$\textsc{mTauL}\frac{m \xrightarrow{\tau} m'}{m\odot n \xrightarrow{\tau} m'\odot n} \qquad \textsc{mVrE}\frac{}{\mathsf{end}\odot\mathsf{end} \xrightarrow{\tau} \mathsf{end}} \qquad \textsc{mVrC1}\frac{}{\mathsf{yes}\otimes m \xrightarrow{\tau} m}$$

$$\textsc{mVrC2}\frac{}{\mathsf{no}\otimes m \xrightarrow{\tau} \mathsf{no}} \qquad \textsc{mVrD1}\frac{}{\mathsf{no}\oplus m \xrightarrow{\tau} m} \qquad \textsc{mVrD2}\frac{}{\mathsf{yes}\oplus m \xrightarrow{\tau} \mathsf{yes}}$$

Fig. 3. Monitor syntax and labelled-transition semantics

For better readability and familiarity, we use LTL for the examples that can be encoded accordingly. Note that since we operate in the finfinite domain, X should be read as a *strong* next operator, in line with Example 3.

The Monitors. We consider the operational monitoring system of [3,30], summarised in Fig. 3 (symmetric rules for binary operators are omitted). The full system is given in [3]. Monitors are states of a transition system where $m+n$ denotes an (external) choice and $m\odot n$ denotes a composite monitor where $\odot \in \{\oplus, \otimes\}$. There are three distinct *verdict* states, yes, no, and end, although only the first two are relevant to monitorability.

This semantics gives an operational account of how a monitor in state m incrementally analyses a sequence of actions $s = a_1 \dots a_k$ to reach a new state n; the monitor m accepts (*resp.*, rejects) a trace f, $\mathbf{acc}(m,f)$ (*resp.*, $\mathbf{rej}(m,f)$), when it can transition to the verdict state yes (*resp.*, no) while analysing a prefix $s \preceq f$ (*i.e.*, s denotes an incomplete trace). Since verdicts are irrevocable (rule mVer in Fig. 3), it is not hard to see that this operational framework satisfies the conditions for a monitoring system of Definition 1. The monitoring system of Fig. 3 is also maximal for regular properties, according to Definition 2. This concrete instance thus demonstrates the realisability of the abstract definitions in Sect. 3.

Theorem 2. *For all $\varphi \in$ recHML, there is a monitor $m \in$ Mon that is sound for φ and accepts all finite traces that positively determine φ and rejects all finite traces that negatively determine φ.*

As a corollary of Theorem 2, from Lemma 1 we deduce that for any arbitrary monitoring system $(M, \mathbf{acc}, \mathbf{rej})$, if $m \in M$ is sound for some $\varphi \in$ recHML, then there is a monitor $n \in$ Mon from Fig. 3 that accepts (*resp.*, rejects) all traces f that m accepts (*resp.*, rejects). In the sequel, we thus assume that the fixed monitoring system is ($\textsc{Mon}, \textbf{acc}, \textbf{rej}$) of Fig. 3, as it subsumes all others.

5 A Syntactic Characterisation of Monitorability

We present syntactic characterizations for the various monitorability classes as fragments of $\textsc{rec}$HML.

Partial Monitorability, Syntactically. In [3], Aceto *et al.* identify a maximal partially monitorable syntactic fragment of $\textsc{rec}$HML.

Theorem 3. (Partially-Complete Monitorability [3]). *Consider the syntactic fragments:*

$$\varphi, \psi \in \textsc{s}\mathrm{HML} ::= \mathsf{tt} \mid \mathsf{ff} \mid [a]\varphi \mid \varphi \wedge \psi \mid \mathsf{max}\, X.\varphi \mid X \text{ and}$$
$$\varphi, \psi \in \textsc{c}\mathrm{HML} ::= \mathsf{tt} \mid \mathsf{ff} \mid \langle a \rangle\varphi \mid \varphi \vee \psi \mid \mathsf{min}\, X.\varphi \mid X.$$

The fragment $\textsc{s}$HML *is monitorable for violation whereas* $\textsc{c}$HML *is monitorable for satisfaction. Furthermore, if* $\varphi \in \textsc{rec}\mathrm{HML}$ *is monitorable for satisfaction (resp., for violation) by some* $m \in \textsc{Mon}$*, it is expressible in* $\textsc{c}$HML*(resp.,* $\textsc{s}$HML*), i.e.,* $\exists \psi \in \textsc{c}\mathrm{HML}$ *(resp.,* $\psi \in \textsc{s}\mathrm{HML}$*), such that* $[\![\varphi]\!] = [\![\psi]\!]$*.*

As a corollary of Theorem 3 we obtain *maximality*: any $\varphi \in \textsc{rec}\mathrm{HML}$ that is monitorable for satisfaction (*resp.*, for violation) can also be expressed as some $\psi \in \textsc{c}\mathrm{HML}$ (*resp.*, $\psi \in \textsc{s}\mathrm{HML}$) where $[\![\varphi]\!] = [\![\psi]\!]$. For this fragment, the following automated synthesis function, which is readily implementable, is given in [3].

$$\begin{array}{rclrclrcl}
\mathsf{m}(\mathsf{ff}) & \overset{\text{def}}{=} & \mathsf{no} & \mathsf{m}(\varphi_1 \wedge \varphi_2) & \overset{\text{def}}{=} & \mathsf{m}(\varphi_1) \otimes \mathsf{m}(\varphi_2) & \mathsf{m}(\mathsf{max}\, X.\varphi) & \overset{\text{def}}{=} & \mathsf{rec}\, x.\mathsf{m}(\varphi) \\
\mathsf{m}(\mathsf{tt}) & \overset{\text{def}}{=} & \mathsf{yes} & \mathsf{m}(\varphi_1 \vee \varphi_2) & \overset{\text{def}}{=} & \mathsf{m}(\varphi_1) \otimes \mathsf{m}(\varphi_2) & \mathsf{m}(\mathsf{min}\, X.\varphi) & \overset{\text{def}}{=} & \mathsf{rec}\, x.\mathsf{m}(\varphi) \\
\mathsf{m}([a]\varphi) & \overset{\text{def}}{=} & \multicolumn{4}{l}{a.\mathsf{m}(\varphi) + \sum_{b \in \textsc{Act} \setminus \{a\}} b.\mathsf{yes}} & \mathsf{m}(X) & \overset{\text{def}}{=} & x \\
\mathsf{m}(\langle a \rangle\varphi) & \overset{\text{def}}{=} & \multicolumn{4}{l}{a.\mathsf{m}(\varphi) + \sum_{b \in \textsc{Act} \setminus \{a\}} b.\mathsf{no}} & & &
\end{array}$$

Informative Monitorability, Syntactically. We proceed to identify syntactic fragments of $\textsc{rec}$HML that correspond to informative monitorability.

Definition 8. *The informative fragment is* $\textsc{i}\mathrm{HML} = \textsc{si}\mathrm{HML} \cup \textsc{ci}\mathrm{HML}$ *where*

$$\textsc{si}\mathrm{HML} = \{\varphi_1 \wedge \varphi_2 \in \textsc{rec}\mathrm{HML} \mid \varphi_1 \in \textsc{s}\mathrm{HML} \textit{ and } \mathsf{ff} \textit{ appears in } \varphi_1\},$$
$$\textsc{ci}\mathrm{HML} = \{\varphi_1 \vee \varphi_2 \in \textsc{rec}\mathrm{HML} \mid \varphi_1 \in \textsc{c}\mathrm{HML} \textit{ and } \mathsf{tt} \textit{ appears in } \varphi_1\} \qquad \blacksquare$$

Theorem 4. *For* $\varphi \in \textsc{rec}\mathrm{HML}$*,* φ *is informatively monitorable if and only if there is some* $\psi \in \textsc{i}\mathrm{HML}$*, such that* $[\![\psi]\!] = [\![\varphi]\!]$*.*

The maximality results of Theorems 3 and 4 permits tool constructions to concentrate on the syntactic fragments identified when synthesising monitors. Theorems 3 and 4 also serve as a lightweight (syntactic) check to determine when a property is monitorable (according to the monitorability classes in Fig. 1).

Example 5. The property φ_{evenW} from Example 4 is monitorable for violation; this can be easily determined since it is expressible in sHML. By contrast, φ_{even} from Example 4 cannot be expressed in either sHML or cHML. In fact, it is *not* partially-complete monitorable: it cannot be satisfaction complete because the trace $(\mathsf{rs})^\omega \in [\![\varphi_{\text{even}}]\!]$ but no prefix can be accepted since they all violate the property; it cannot be violation complete either, since the trace $\epsilon \notin [\![\varphi_{\text{even}}]\!]$ but is can be extended by $(\mathsf{rs})^\omega$ which makes (persistent) rejection verdicts unsound. The property $\mathsf{G}\,\neg\mathsf{f} \wedge \mathsf{F}\,\mathsf{s}$ from Example 2 (expressed here in LTL) is a siHML property, as $\mathsf{G}\neg\mathsf{f}$ can be written in sHML as $\max X.[\mathsf{f}]\mathsf{ff}\wedge[\mathsf{s}]X\wedge[\mathsf{r}]X$. In contrast, $\mathsf{FG}\neg\mathsf{r}$ cannot be written in iHMLsince it is not informatively monitorable. ■

Remark 2. In siHML and ciHML, φ_1 describes an informative part of the formula, that is, a formula with at least one path to tt (or ff), which indicates that the corresponding finite trace determines the property. Monitor synthesis from these fragments can use this part of the formula to synthesize a monitor that detects the finite traces that satisfy (violate) φ_1. The value of the synthesised monitor then depends on φ_1. It is therefore important to have techniques to extract some φ_1 that will retain as much monitoring information as possible. This extraction is outside the scope of this paper and left as future work. ■

Persistently Informative Monitorability, Syntactically. We also give a syntactic characterization of the recHML properties that are informatively monitorable for satisfaction or violation. As the requirements for persistently informative monitors are subtler than for informative monitors, the fragments we present are more involved than those for informative monitorability.

Definition 9. *We define* eHML*, the explicit fragment of* recHML*:*

$$\begin{array}{rlllll} \varphi \in \text{eHML} ::= \mathsf{tt} & \mid \mathsf{ff} & \mid \min X.\varphi & \mid \max X.\varphi & \mid X \\ \mid \varphi \vee \psi & \mid \varphi \wedge \psi & \mid \bigvee_{\alpha\in\text{Act}} \langle\alpha\rangle\varphi_\alpha & \mid \bigwedge_{\alpha\in\text{Act}} [\alpha]\varphi_\alpha. & \end{array}$$
■

Example 6. Formula $[\mathsf{f}][\mathsf{s}]\mathsf{ff}$ is not explicit, but, assuming that $\text{Act} = \{\mathsf{f},\mathsf{s},\mathsf{r}\}$, it can be rewritten as the explicit formula $[\mathsf{f}]([\mathsf{s}]\mathsf{ff} \wedge [\mathsf{f}]\mathsf{tt} \wedge [\mathsf{r}]\mathsf{tt}) \wedge [\mathsf{s}]\mathsf{tt} \wedge [\mathsf{r}]\mathsf{tt}$. ■

Roughly, the following definition captures whether tt and ff are reachable from subformulae (where the binding of a variable is reachable from the variable).

Definition 10. *Given a closed* sHML *(resp.,* cHML*) formula φ, we define for a subformula ψ that it can refute (resp., verify) in 0 unfoldings, when* ff *(resp.,* tt*) appears in ψ, and that it can refute (resp., verify) in $k+1$ unfoldings, when it can refute (resp., verify) in k unfoldings, or X appears in ψ and ψ is in the scope of a subformula* $\max X.\psi'$ *(resp.,* $\min X.\psi'$*) that can refute (resp., verify) in k unfoldings. We simply say that ψ can refute (resp., verify) when it can refute (resp., verify) in k unfoldings, for some $k \geq 0$.* ■

Example 7. For formula $\mathsf{max}\,X.[\mathsf{s}]X \wedge [\mathsf{f}]\mathsf{ff} \wedge [\mathsf{r}]\mathsf{ff}$, subformula $[\mathsf{s}]X \wedge [\mathsf{f}]\mathsf{ff} \wedge [\mathsf{r}]\mathsf{ff}$ can refute in 0 unfoldings. In contrast, $[\mathsf{s}]X$ cannot refute in 0 unfoldings, but it can refute in 1, because X appears in it and $\mathsf{max}\,X.[\mathsf{s}]X \wedge [\mathsf{f}]\mathsf{ff} \wedge [\mathsf{r}]\mathsf{ff}$ can refute in 0 unfoldings. Therefore, all subformulae of $\mathsf{max}\,X.[\mathsf{s}]X \wedge [\mathsf{f}]\mathsf{ff} \wedge [\mathsf{r}]\mathsf{ff}$ can refute. ■

We now define the fragments of recHML corresponding to recHML properties that are persistently informatively monitorable for satisfaction or violation.

Definition 11. *We define the fragment* pHML = spHML ∪ cpHML *where:*

$$\mathrm{spHML} = \left\{ \varphi_1 \wedge \varphi_2 \in \mathrm{recHML} \;\middle|\; \begin{array}{l} \varphi_1 \in \mathrm{sHML} \cap \mathrm{eHML} \textit{ and every} \\ \textit{subformula of } \varphi_1 \textit{ can refute} \end{array} \right\}$$

$$\mathrm{cpHML} = \left\{ \varphi_1 \vee \varphi_2 \in \mathrm{recHML} \;\middle|\; \begin{array}{l} \varphi_1 \in \mathrm{cHML} \cap \mathrm{eHML} \textit{ and every} \\ \textit{subformula of } \varphi_1 \textit{ can verify} \end{array} \right\} \quad ■$$

Theorem 5. *For $\varphi \in$ recHML, φ is persistently informatively monitorable for violation (resp., for satisfaction) if and only if there is some $\psi \in$ spHML (resp., $\psi \in$ cpHML), such that $[\![\psi]\!] = [\![\varphi]\!]$.*

Remark 3. To the best of our efforts, a syntactic characterisation of persistently informative monitorability would involve pairs of equivalent formulae with parts from sHML and cHML that together become, in some sense, explicit. We leave such a characterization as future work. ■

6 Safety and Co-safety

The classic (and perhaps the most intuitive) definition of monitorability consists of (some variation of) *safety* properties [3,7,25,32,44,45]. There are, however, subtleties associated with how exactly safety properties are defined—particularly over the finfinite domain—and how decidable they need to be to qualify as truly monitorable. For example, Kim and Viswanathan [45] argued that only recursively enumerable safety properties are monitorable (they restrict themselves to infinite, rather than finfinite traces). By and large, however, most works on monitorability restrict themselves to regular properties, as we do in Sect. 4.

We adopt the definition of safety that is intuitive for the context of RV: a property can be considered monitorable if its failures can be identified by a finite prefix. This is equivalent to Falcone *et al.*'s definition of safety properties [25, Def. 4] and, when restricted to infinite traces, to other work such as [7,16,32].

Definition 12 (Safety). *A property $P \subseteq \mathrm{Act}^\infty$ is a* safety property *if every $f \notin P$ has a prefix that determines P negatively. The class of safety properties is denoted as* Safe *in Fig. 1.* ■

Pnueli and Zaks, and Falcone *et al.* (among others) argue that it makes sense to monitor both for violation and satisfaction.Hence, if safety is monitorable for violations, then the dual class, co-safety (*a.k.a.* guarantee [25], reachability [15]), is monitorable for satisfaction. That is, every trace that satisfies a co-safety property can be positively determined by a finite prefix.

Definition 13 (Co-safety). *A property $P \subseteq \textsc{Act}^\infty$ is a* co-safety property *if every $f \in P$ has prefix that determines P positively. The class of co-safety properties is denoted as CoSafe, also represented in Fig. 1.* ■

Example 8. "Eventually s *is reached", i.e.,* $\mathsf{F\,s}$, is a co-safety property whereas "f *never occurs", i.e.,* $\mathsf{G}\,\neg\mathsf{f}$, is a safety property. The property "s *occurs infinitely often", i.e.,* $\mathsf{G\,F\,s}$, is neither safety nor co-safety. The property only holds over infinite traces so it cannot be positively determined by a finite trace. Dually, there is *no* finite trace that determines that there cannot be an infinite number of s occurrences in a continuation of the trace. Similarly, φ_{even} from Example 4 is neither a safety nor a co-safety property, but φ_{evenW} is a safety property. ■

Safety and Co-safety, Operationally. It should come as no surprise that safety and co-safety coincide with an equally natural operational definition.Here, we establish the correspondence with the denotational definition of safety (co-safety), completing three correspondences amongst the monitorability classes of Fig. 1.

Theorem 6. *$\mathsf{VCmp} = \mathsf{Safe}$ and $\mathsf{SCmp} = \mathsf{CoSafe}$.*

Proof. We treat the case for safety, as the case for co-safety is similar. If P is a safety property, then for every $f \in \textsc{Act}^\infty \setminus P$, there is some finite prefix s of f that negatively determines P. Therefore, m_P is sound (Lemma 2) and violation-complete (Definition 2) for P. The other direction follows from the fact that whenever $P \subseteq \textsc{Act}^\infty$ is monitorable for violation, every $f \in \textsc{Act}^\infty \setminus P$ has a finite prefix that negatively determines it.

Aceto *et al.* [3] already show the correspondence between violation (dually, satisfaction) monitorability over finfinite traces and properties expressible in sHML (dually, cHML). As a corollary of Theorem 6, we obtain a syntactic characterisation for the Safe and CoSafe monitorability classes; see Fig. 1.

Remark 4. Falcone *et al.* [25, Def. 17, Thm. 3] propose definitions of monitorability over finfinite traces that are claimed to coincide with the classes Safe, CoSafe and their union. However, this claim is incorrect. The properties, "the trace is finite" and $\mathsf{G\,F\,s}$ from Example 8 are neither safety nor co-safety properties. On the other hand, they are monitorable according to the alternative monitorability definition given in [25, Def. 17]. If the results claimed in [25, Thm. 3] held true, this would contradict the fact that those properties are neither safety nor co-safety properties. See [4] for further details. ■

7 Pnueli and Zaks

The work on monitorability due to Pnueli and Zaks [41] is often cited by the RV community [12]. The often overlooked particularity of their definitions is that they only define monitorability of a property *with respect to a (finite) sequence.*

Definition 14. ([41]). *Property P is s-monitorable, where $s \in \textsc{Act}^*$, if there is some $r \in \textsc{Act}^*$ such that P is positively or negatively determined by sr.* ■

Example 9. The property $(\mathsf{f} \wedge \mathsf{F}\,\mathsf{r}) \vee (\mathsf{F}\,\mathsf{G}\,\mathsf{s})$ is s-monitorable for any finite trace that begins with f, *i.e.,* $\mathsf{f}s$, since it is determined by the extension $\mathsf{f}s\mathsf{r}$. It is *not* s-monitorable for finite traces that begin with an action other than f. ■

Monitorability over properties—rather than over property–sequence pairs—can then be defined by either quantifying *universally* or *existentially* over finite traces: a property is monitorable either if it is s-monitorable for all s, or for some s. We address both definitions, which we call ∀PZ- and ∃PZ-monitorability respectively. ∀PZ-monitorability is the more standard interpretation: it appears for example in [13,25] where it is attributed to Pnueli and Zaks. However, the original intent seems to align more with ∃PZ-monitorability: in [41], Pnueli and Zaks refer to a property as non-monitorable if it is not monitorable for *any* sequence. This interpretation coincides with *weak monitorability* used in [18].

Definition 15. (∀PZ-monitorability). *A property P is (universally Pnueli–Zaks) ∀PZ-monitorable if it is s-monitorable for* all *finite traces s. The class of all ∀PZ-monitorable properties is denoted $\forall\mathsf{PZ}$.* ■

Definition 16. (∃PZ-monitorability). *A property is (existentially Pnueli–Zaks) ∃PZ-monitorable if it is s-monitorable for* some *finite trace s, i.e., if it is ε-monitorable. The class of ∃PZ-monitorable properties is written $\exists\mathsf{PZ}$.* ■

The apparently innocuous choice between existential and universal quantification leads to different monitorability classes $\forall\mathsf{PZ}$ and $\exists\mathsf{PZ}$.

Example 10. Consider the property *"Either s occurs before f, or r happens infinitely often"*, expressed in LTL fashion as $((\neg\mathsf{f})\,\mathsf{U}\,\mathsf{s}) \vee (\mathsf{G}\,\mathsf{F}\,\mathsf{r})$. This property is ∃PZ-monitorable because the trace s positively determines the property. However, it is *not* ∀PZ-monitorable because no extension of the trace f positively or negatively determines that property. Indeed, all extensions of f violate the first disjunct and, as we argued in Example 8, there is no finite trace that determines the second conjunct positively or negatively. Property φ_{even} from Example 4 is ∀PZ-monitorable: any prefix of the form $a_0\mathsf{s}\ldots a_n\mathsf{s}$ or $a_0\mathsf{s}\ldots a_n$ (including ϵ), where $n \geq 0$ and every $a_i \in \{\mathsf{s}, \mathsf{f}, \mathsf{r}\}$, can be extended to a prefix that negatively determines it (*e.g.,* by extending it with ff). ■

From Definitions 15 and 16, it follows immediately that $\forall\mathsf{PZ} \subset \exists\mathsf{PZ}$.

Proposition 2. *All properties in $\mathsf{Safe} \cup \mathsf{CoSafe}$ are ∀PZ-monitorable.*

Proof. Let $P \in \mathsf{Safe}$ and pick a finite trace s. If there is an f such that $sf \notin P$ then, by Definition 12, there exists $r \preceq sf$ that negatively determines P, meaning that s has an extension that negatively determines P. Alternatively, if there is *no* f such that $sf \notin P$, s itself positively determines P. Hence P is s-monitorable, for *every* s, according to Definition 14. The case for $P \in \mathsf{CoSafe}$ is dual. □

Pnueli and Zaks, Operationally. ∃PZ-monitorability coincides with informative monitorability: ∃PZ-monitorable properties are those for which some monitor can reach a verdict on some finite trace. For similar reasons, ∀PZ-monitorability coincides with informative monitorability. See Fig. 1.

Theorem 7. *$\exists\mathsf{PZ} = \mathsf{ICmp}$ and $\forall\mathsf{PZ} = \mathsf{PICmp}$.*

Proof. Since the proofs of the two claims are analogous, we simply outline the one for $\forall\mathsf{PZ} = \mathsf{PICmp}$. Let $P \in \forall\mathsf{PZ}$ and pick a finite trace $s \in \text{ACT}^*$. By Lemma 2, m_P is sound for P. By Definition 6 we need to show that there exists an f such that $\mathbf{acc}(m_P, sf)$ or $\mathbf{rej}(m_P, sf)$. From Definition 15 and 14 we know that there is a finite r such that sr positively or negatively determines P. By Definition 2 we know that $\mathbf{acc}(m_P, sr)$ or $\mathbf{rej}(m_P, sr)$. Thus $P \in \mathsf{PICmp}$, which is the required result.

Conversely, assume $P \in \mathsf{PICmp}$, and pick some $s \in \text{ACT}^*$. By Definitions 15 and 14, we need to show that there is an extension of s that positively or negatively determines P. From Definitions 6 and 7, there exists some f such that $\mathbf{acc}(m_P, sf)$ or $\mathbf{rej}(m_P, sf)$. By Definition 1, there is a finite extension of s, say sr, that is a prefix of sf such that $\mathbf{acc}(m_P, sr)$ or $\mathbf{rej}(m_P, sr)$. By Definition 2, we know that sr either positively or negatively determines P.Thus $P \in \forall\mathsf{PZ}$.

8 Monitorability in Other Settings

We have shown how classical definitions of monitorability fit into our hierarchy and provided the corresponding operational interpretations and syntactic characterisations, focusing on regular finfinite properties over a finite alphabet and monitors with irrevocable verdicts. Here we discuss how different parameters, both within our setting and beyond, affect what is monitorable.

Monitorability with respect to the Alphabet. The monitorability of a property can depend on ACT. For instance, if ACT has at least two elements $\{a, b, \ldots\}$, property $\{a^\omega\}$, which can be represented as $\mathsf{max}X.\langle a\rangle X$, is s-monitorable for every sequence s, as s can be extended to sb, which negatively determines the property. On the other hand, assume that ACT $= \{a\}$. In this case, $\{a^\omega\}$ is neither $\exists$PZ- nor $\forall$PZ-monitorable. Indeed, no string $s = a^k$, $k \geq 0$, determines $\{a^\omega\}$ positively or negatively as s does not satisfy it but its extension a^ω does. On the other hand, when restricted to infinite traces, $\{a^\omega\}$ is again $\exists$PZ-monitorable.

So far, we only considered finite alphabets; how an infinite alphabet, which may encode integer data for example, affects monitorability is left as future work.

Monitoring with Revocable Verdicts. Early on, we postulated that verdicts are irrevocable. Although this is a typical (implicit) assumption in most work on monitorability, some authors have considered monitors that give revocable judgements when an irrevocable one is not appropriate. This approach is taken by Bauer *et al.* when they define a finite-trace semantics for LTL, called RV-LTL [13]. Falcone *et al.* [25] also have a definition of monitorability based on this idea (in addition to those discussed in Remark 4). It uses the four-valued domain $\{\mathsf{yes}, \mathsf{no}, \mathsf{yes}_c, \mathsf{no}_c\}$ (c for *currently*). Finite traces that do not determine a property yield a (revocable) verdict yes_c or no_c that indicates whether the trace observed so far satisfies the property; yes and no are still irrevocable. This definition allows *all* finfinite properties to be monitored since it does not require verdicts to be irrevocable.

This type of monitoring does not give any guarantees beyond soundness: there are properties that are monitorable according to this definition for which no sound monitor ever reaches an irrevocable verdict: $\mathsf{F}\ \mathsf{G}\ \mathsf{s}$ for the system from Example 1 has no sound informative monitor, yet can be monitored according to Falcone *et al.* 's four-valued monitoring. This type of monitorability is complete, in the sense of providing at least a *revocable* verdict for all traces.

Monitorability in the Infinite and Finite. Bauer *et al.* use $\forall$PZ-monitorability in their study of runtime verification for LTL [14] and attribute it to Pnueli and Zaks. However, unlike Falcone *et al.*, Pnueli and Zaks [41] and ourselves, they focus on properties over *infinite* traces. There are some striking differences that arise if there is no risk of an execution ending. Aceto *et al.* show that, unlike in the finfinite domain, a set of non-trivial properties becomes completely monitorable: HML [33] (*a.k.a.* modal logic) is both satisfaction- and violation-monitorable over infinite traces [3]. Furthermore, some properties, like $\{a^\omega\}$ over $\textsc{Act} = \{a\}$, that were not $\exists$PZ- or $\forall$PZ-monitorable on the finfinite domain, are $\exists$PZ- or even $\forall$PZ-monitorable on the infinite domain. The full analysis of how the hierarchy in Fig. 1 changes for the infinite domain is left for future work.

Havelund and Peled recently presented a related classification of infinitary properties [32]. Their classification consists of safety and co-safety properties, (there called AFS and AFR), and properties that are not positively or not negatively determined by any sequence (NFS and NFR) and properties where some, but not all prefixes have an extention that determines the property positively, and their negations (SFS and SFR). They show that several of their classes contain both $\forall$PZ-monitorable and non-$\forall$PZ-monitorable properties. In contrast, in our classification, $\forall$PZ-monitorability is not orthogonal to other types of monitorability; rather, it is part of a spectrum that reflects the trade-offs between the strengths of the guarantees a monitor can provide and the specifications that can be monitored with these guarantees.

Barringer *et al.* [11] consider monitoring of properties over *finite* traces. In this domain, all properties are monitorable if, as is the case in [11], the end of a trace is *observable*; in this setting the question of monitorability is less relevant.

Monitoring Non-regular Properties. Although we have focussed on the monitorability of regular properties, the monitorability hierarchy of Sect. 3 is not restricted to this setting. Indeed, although non-regular properties require richer monitors, for example monitors with a stack or registers, the same concerns of soundness and degress of completeness remain relevant. Barringer *et al.* consider a specification logic that allows for context-free properties [11]. In [26], Ferrier *et al.* consider monitors with registers (*i.e.,* infinite state monitors) to verify safety properties that are not regular. Characterising (*e.g.,* syntactically) the different classes of monitorability for non-regular properties is left as future work.

Beyond Monitorability. Stream-based monitoring systems such as [21,22] are more concerned with producing (revocable) aggregate outputs and transforming traces to satisfy properties, employing more powerful monitors than the ones considered here (*e.g.,* transducers). Instead of monitorability, *enforceability* [5, 25] is a criterion that is better suited for these settings.

9 Conclusion

We have proposed a unified, operational view on monitorability. This allows us to clearly state the implicit operational guarantees of existing definitions of monitorability. For instance, recall Example 1 from the introduction: since $(\mathsf{G}\,\neg\mathsf{f}) \wedge (\mathsf{F}\,\mathsf{s})$ is $\exists$PZ- and $\forall$PZ-monitorable but it is neither a safety nor a co-safety property, we know there is a monitor which can recognise some violations and satisfactions of this property, but there is no monitor that can recognise *all* satisfactions or *all* violations. Although we focussed on regular, finfinite properties, the definitions of monitorability in Sect. 3, and, more fundamentally, the methodology that systematically puts the relationship between monitor behaviour and specification centre stage, are equally applicable to other settings.

The emphasis our approach places on the explicit guarantees provided by the different types of monitorability should clarify the role of monitorability in the design of RV tools which, depending on the setting, may have different requirements. Indeed, a monitor that checks that the output of a module does not violate the preconditions of the next module had better be violation-complete; on the other hand, it is probably sufficient that a monitor be informative when it is used as a light-weight, best-effort part of a hybrid verification strategy.

References

1. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A.: Monitoring for silent actions. In: Lokam, S., Ramanujam, R. (eds.) FSTTCS. LIPIcs, vol. 93, pp. 7:1–7:14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2017)
2. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A.: A framework for parameterized monitorability. In: Baier, C., Dal Lago, U. (eds.) FoSSaCS 2018. LNCS, vol. 10803, pp. 203–220. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89366-2_11
3. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Lehtinen, K.: Adventures in monitorability: from branching to linear time and back again. Proc. ACM Program. Lang. **3**(POPL), 52:1–52:29 (2019). https://dl.acm.org/citation.cfm?id=3290365
4. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Lehtinen, K.: An operational guide to monitorability. CoRR abs/1906.00766 (2019). http://arxiv.org/abs/1906.00766
5. Aceto, L., Cassar, I., Francalanza, A., Ingólfsdóttir, A.: On runtime enforcement via suppressions. In: 29th International Conference on Concurrency Theory, CONCUR 2018. LIPIcs, vol. 118, pp. 34:1–34:17. Schloss Dagstuhl (2018). https://doi.org/10.4230/LIPIcs.CONCUR.2018.34

6. Aceto, L., Ingólfsdóttir, A., Larsen, K.G., Srba, J.: Reactive Systems: Modelling, Specification and Verification. Cambridge University Press, New York (2007)
7. Alpern, B., Schneider, F.B.: Defining liveness. Inf. Process. Lett. **21**(4), 181–185 (1985)
8. Attard, D.P., Cassar, I., Francalanza, A., Aceto, L., Ingolfsdottir, A.: A runtime monitoring tool for actor-based systems. In: Gay, S., Ravara, A. (eds.) Behavioural Types: From Theory to Tools, pp. 49–74. River Publishers (2017)
9. Attard, D.P., Francalanza, A.: A monitoring tool for a branching-time logic. In: Falcone, Y., Sánchez, C. (eds.) RV 2016. LNCS, vol. 10012, pp. 473–481. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46982-9_31
10. Baier, C., Tinelli, C. (eds.): Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, LNCS, vol. 9035. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0
11. Barringer, H., Rydeheard, D., Havelund, K.: Rule systems for run-time monitoring: from Eagle to RuleR. J. Log. Comput. **20**(3), 675–706 (2008)
12. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to runtime verification. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification. LNCS, vol. 10457, pp. 1–33. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5_1
13. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. J. Log. Comput. **20**(3), 651–674 (2010)
14. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. **20**(4), 14:1–14:64 (2011). https://doi.org/10.1145/2000799.2000800
15. Bérard, B., et al.: Systems and Software Verification: Model-checking Techniques and Tools. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-662-04558-9
16. Chang, E., Manna, Z., Pnueli, A.: Characterization of temporal property classes. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 474–486. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55719-9_97
17. Chen, F., Rosu, G.: Mop: an efficient and generic runtime verification framework. In: Gabriel, R.P., Bacon, D.F., Lopes, C.V., Steele Jr., G.L. (eds.) Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2007, pp. 569–588. ACM (2007). https://doi.org/10.1145/1297027.1297069
18. Chen, Z., Wu, Y., Wei, O., Sheng, B.: Poster: deciding weak monitorability for runtime verification. In: 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), pp. 163–164, May 2018
19. Cini, C., Francalanza, A.: An LTL proof system for runtime verification. In: Baier and Tinelli [10], pp. 581–595. https://doi.org/10.1007/978-3-662-46681-0_54
20. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT press (1999)
21. Convent, L., Hungerecker, S., Leucker, M., Scheffel, T., Schmitz, M., Thoma, D.: TeSSLa: temporal stream-based specification language. In: Massoni, T., Mousavi, M.R. (eds.) SBMF 2018. LNCS, vol. 11254, pp. 144–162. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03044-5_10
22. D'Angelo, B., et al.: LOLA: runtime monitoring of synchronous systems. In: 12th International Symposium on Temporal Representation and Reasoning (TIME 2005), pp. 166–174. IEEE Computer Society Press, June 2005
23. Decker, N., Leucker, M., Thoma, D.: jUnit$^{\mathrm{RV}}$–adding runtime verification to jUnit. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 459–464. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38088-4_34

24. Diekert, V., Leucker, M.: Topology, monitorable properties and runtime verification. Theor. Comput. Sci. **537**, 29–41 (2014). https://doi.org/10.1016/j.tcs.2014.02.052
25. Falcone, Y., Fernandez, J.C., Mounier, L.: What can you verify and enforce at runtime? Int. J. Softw. Tools Technol. Transf. **14**(3), 349–382 (2012)
26. Ferrère, T., Henzinger, T.A., Saraç, N.E.: A theory of register monitors. In: Dawar, A., Grädel, E. (eds.) Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, pp. 394–403. ACM (2018). https://doi.org/10.1145/3209108.3209194
27. Francalanza, A.: A theory of monitors (extended abstract). In: Jacobs, B., Löding, C. (eds.) FoSSaCS 2016. LNCS, vol. 9634, pp. 145–161. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49630-5_9
28. Francalanza, A.: Consistently-detecting monitors. In: 28th International Conference on Concurrency Theory (CONCUR). LIPIcs, vol. 85, pp. 8:1–8:19. Schloss Dagstuhl (2017). https://doi.org/10.4230/LIPIcs.CONCUR.2017.8
29. Francalanza, A., et al.: A foundation for runtime monitoring. In: Lahiri, S., Reger, G. (eds.) RV 2017. LNCS, vol. 10548, pp. 8–29. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67531-2_2
30. Francalanza, A., Aceto, L., Ingólfsdóttir, A.: Monitorability for the Hennessy-Milner logic with recursion. Form. Methods Syst. Des. **51**(1), 87–116 (2017). https://doi.org/10.1007/s10703-017-0273-z
31. Francalanza, A., Seychell, A.: Synthesising correct concurrent runtime monitors. Form. Methods Syst. Des. (FMSD) **46**(3), 226–261 (2015). https://doi.org/10.1007/s10703-014-0217-9
32. Havelund, K., Peled, D.: Runtime verification: from propositional to first-order temporal logic. In: Colombo, C., Leucker, M. (eds.) RV 2018. LNCS, vol. 11237, pp. 90–112. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03769-7_7
33. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. J. ACM **32**(1), 137–161 (1985). https://doi.org/10.1145/2455.2460
34. Kozen, D.C.: Results on the propositional μ-calculus. Theor. Comput. Sci. **27**, 333–354 (1983)
35. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Form. Methods Syst. Des. **19**(3), 291–314 (2001)
36. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. J. ACM **47**(2), 312–360 (2000)
37. Larsen, K.G.: Proof systems for satisfiability in Hennessy-Milner logic with recursion. Theor. Comput. Sci. **72**(2), 265–288 (1990). https://doi.org/10.1016/0304-3975(90)90038-J
38. Laurent, J., Goodloe, A., Pike, L.: Assuring the guardians. In: Bartocci, E., Majumdar, R. (eds.) RV 2015. LNCS, vol. 9333, pp. 87–101. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23820-3_6
39. Manna, Z., Pnueli, A.: Completing the temporal picture. Theor. Comput. Sci. **83**(1), 97–130 (1991). https://doi.org/10.1016/0304-3975(91)90041-Y
40. Neykova, R., Bocchi, L., Yoshida, N.: Timed runtime monitoring for multiparty conversations. Form. Asp. Comput. **29**(5), 877–910 (2017). https://doi.org/10.1007/s00165-017-0420-8
41. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 573–586. Springer, Heidelberg (2006). https://doi.org/10.1007/11813040_38

42. Reger, G., Cruz, H.C., Rydeheard, D.E.: MarQ: monitoring at runtime withQEA. In: Baier and Tinelli [10], pp. 596–610. https://doi.org/10.1007/978-3-662-46681-0_55
43. Rosu, G.: On safety properties and their monitoring. Sci. Ann. Comput. Sci. **22**(2), 327–365 (2012)
44. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. **3**(1), 30–50 (2000)
45. Viswanathan, M., Kim, M.: Foundations for the run-time monitoring of reactive systems – *Fundamentals of the MaC Language*. In: Liu, Z., Araki, K. (eds.) ICTAC 2004. LNCS, vol. 3407, pp. 543–556. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31862-0_38
46. Wolper, P.: Temporal logic can be more expressive. Inf. Control **56**(1/2), 72–99 (1983). https://doi.org/10.1016/S0019-9958(83)80051-5