Noname manuscript No.
(will be inserted by the editor)

An Operational Guide to Monitorability

Luca Aceto · Antonis Achilleos · Adrian Francalanza · Anna Ingólfsdóttir · Karoliina Lehtinen

Received: date / Accepted: date

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.

Keywords Runtime Verification · Monitors · Monitorability · Logical Fragments

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), the EPSRC project "Solving parity games in theory and practice" (№:EP/P020909/1), project BehAPI, funded by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement №:778233, and the Italian MIUR project PRIN 2017FTXR7S IT MATTERS "Methods and Tools for Trustworthy Smart Systems".

L. Aceto
Gran Sasso Science Institute, L'Aquila, Italy

L. Aceto · A. Achilleos · Ingólfsdóttir
Reykjavik University, Reykjavik, Iceland

A. Francalanza
University of Malta, Msida, Malta

K. Lehtinen
University of Liverpool, Liverpool, UK

1 Introduction

Runtime Verification (RV) [15] 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, 5, 22, 29, 35, 45, 47]. 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* [15, 47] concerns itself 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 [12, 20, 27, 46, 48], 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 to the construction of RV tools: It defines, either explicitly or implicitly, a notion of *monitor correctness* [32, 33, 36, 44], 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, 5, 35], and used as a *core calculus* for studying optimisations of the synthesis algorithm. More broadly, monitorability boundaries may assist in the design of the monitoring set-up, and 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). We therefore emphasize the separation of concerns between the specification of a correctness property on the one hand, and the method(s) used to verify it on the other [35].

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 [5,17,30,35,37,47]. 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 by Falcone *et al.* [30] contain serious errors.

Example 1.1 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 [47] (discussed in Sec. 7), this property is monitorable. However, it is not monitorable according to others, including Schneider [50], Viswanathan and Kim [53], and Aceto *et al.* [5], whose definition of monitorability coincides with some subset of *safety properties.* ■

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 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. Yet, this is key in designing a monitoring set-up. For example, if a monitor is used to check that the input of a critical component, produced by an untrusted third-party component, satisfies some boundary conditions, then it is important that *all* violations are identified. On the other hand, if runtime monitoring is used as a best-effort attempt to catch bugs without model checking, then weaker guarantees can suffice.

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. 1.1 (middle)—which classifies properties according to what types of guarantees RV can give for them. At one extreme, anything can be monitored if the only requirement is for monitors to be *sound*, that is, their verdicts 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. Informatively

[1] In RV, it is commonplace to see the expected monitor behaviour described via an intermediary n-valued logic semantics [16,17,37] (*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.

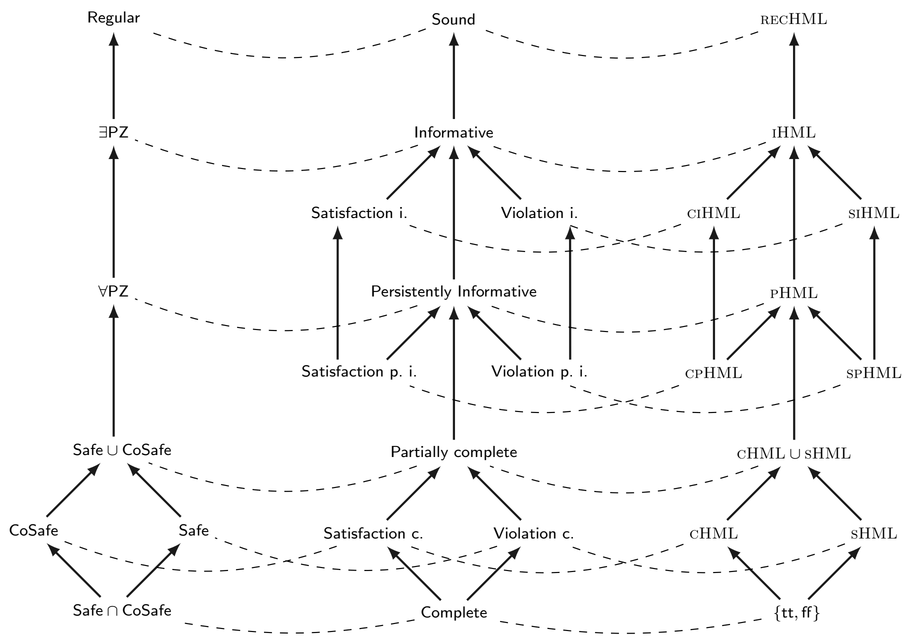

Fig. 1.1 The Monitorability Hierarchy of Regular Properties

monitorable properties can be further categorised into those informatively monitorable for violations and those informatively monitorable for satisfaction. 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*. Requiring a specific verdict to always be reachable further refines this class into into persistently rejecting and persistently accepting 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.

Our first contribution is to define this hierarchy of monitorability, depicted in Fig. 1.1 (middle). In order not to favour a specific operational model, the hierarchy is cast in terms of abstract behavioural requirements for monitors, and is not restricted to regular properties. 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 [5], 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 (Thm. 4.4) 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. Our second contribution is to provide monitorability characterisations as fragments of recHML [8, 43] (a variant of the modal μ-calculus [40]) interpreted over finfinite traces—see Fig. 1.1 (right). This 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 [5]; related synthesis functions based on syntactic characterisations for a branching-time setting [34, 35] have already been implemented in a tool [11, 12]. Here, we provide the missing syntactic characterisations for informative and persistently informative monitorability as well as their violation and satisfaction refinements.[2] Note that we work in the finfinite domain, where executions can be finite or infinite, like Falcone *et al.* did in [30]. This setting is a natural one when it comes to monitoring, as it does not make the potentially unrealistic assumption that executions never stall, deadlock, or otherwise remain silent with respect to the events that are monitored. This gives our result more generality than restricting ourselves to the infinite domain.

Finally, we show that the proposed hierarchy accounts for existing notions of monitorability. See Fig. 1.1 (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.* [30], 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 Thms. 3.1, 6.1 and 7.1;
2. An extension of the syntactic characterisations of monitorable classes from [5], mapping all of these classes to fragments in recHML, which can be viewed as a target byte-code for higher-level logics, see Thms. 5.2 and 5.3.

This article extends the conference version [6]. The main technical novelty here is the logical characterisation of persistently informative monitorability. Furthermore, we refine the monitorability hierarchy by treating informative monitorability and persistently informative monitorability for satisfaction and violation as monitorability classes in their own right (with corresponding logical characterisations). Furthermore, we have added detailed proofs, extended examples and improved explanations.

Roadmap. We start with defining notation for traces and properties in the finfinite domain in Sec. 2. We then define the monitorability hierarchy for properties over finfinite traces in Sec. 3, and instanciate it with concrete operational semantics in Sec. 4 for regular properties. In Sec. 5 we give syntactic characterisations of each level of our hierarchy. In Secs. 6 and 7 we show how existing notions of monitorability embed into our hierarchy and discuss a serious error in Falcone *et al.*'s notion of monitorability. Finally, before concluding, in Sec. 8 we discuss other notions of monitorability and how changing various aspects of the framework, such as

[2] We not that, as depicted in Fig. 1.1, partial monitorability does not imply any of these refinements.

the trace domain or the definition of monitors affects the resulting monitorability hierarchy.

2 Preliminaries

Traces. We assume a *finite* set of actions, $a, b, \ldots \in \textsc{Act}$. The metavariables $t, u \in \textsc{Act}^\omega$ range over *infinite* sequences of actions. *Finite traces*, denoted as $s, r \in \textsc{Act}^*$, represent *finite* prefixes of system runs. We also find it useful to denote sets of finite traces, $S \subseteq \textsc{Act}^*$. Collectively, finite and infinite traces in the set $\textsc{Act}^\infty = \textsc{Act}^\omega \cup \textsc{Act}^*$ are called *finfinite* traces. We use $f, g \in \textsc{Act}^\infty$ to range over finfinite traces and $F \subseteq \textsc{Act}^\infty$ 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.*, there is a g such that $f = sg$. We use the notation $f[k]$ to denote the action at position k in f: for $f = ag$, $f[0] = a$, and for $k \geq 0$, $f[k+1] = g[k]$.

Properties. A *property* over finfinite (*resp.*, infinite) traces, denoted by the variable P, is a subset of $\textsc{Act}^\infty$ (*resp.*, of $\textsc{Act}^\omega$). In general, a *property* refers to a finfinite property, unless stated otherwise. A finite trace s *positively determines* a property $P \subseteq \textsc{Act}^\infty$ when $sf \in P$ for *every* continuation $f \in \textsc{Act}^\infty$; analogously, s *negatively determines* P when $sf \notin P$ for every $f \in \textsc{Act}^\infty$. The same terms apply similarly when $P \subseteq \textsc{Act}^\omega$. We say that P is suffix-closed when for all $s, r \in \textsc{Act}^*$, $s \in P$ implies $sr \in P$ — notice that we only quantify over finite traces. For a given $P \subseteq \textsc{Act}^\infty$ we identify the following two sets of finite traces:

$$D_P^- = \{\, s \in \textsc{Act}^* \mid s \text{ negatively determines } P \,\};$$
$$D_P^+ = \{\, s \in \textsc{Act}^* \mid s \text{ positively determines } P \,\}.$$

We say that a finfinite property is *regular* if it is the union of a regular property $P_{\text{fin}} \subseteq \textsc{Act}^*$ and an ω-regular property $P_{\text{inf}} \subseteq \textsc{Act}^\omega$ [52].

Example 2.1 Recall the system discussed in Example 1.1 with actions failure (f), success (s) and recovery (r). A trace that contains at least two occurrences of r positively determines the property described by the LTL syntax $\mathsf{F}\big(\mathsf{r} \wedge \mathsf{X}(\mathsf{F}\,\mathsf{r})\big)$. A finite trace that contain the action s negatively determines the property $\mathsf{G}\,(\mathsf{f} \vee \mathsf{r}) \wedge \mathsf{F}\,\mathsf{r}$. Note, however, that not all violating traces have a prefix that contains the action s. Indeed, the infinite f^ω does not satisfy this property, but none of its prefixes contain s. ■

3 A Monitor-Oriented Hierarchy

From a tool-construction perspective, it is important to give concrete, implementable definitions of monitors; we do so in Sec. 4. To understand the guarantees that these monitors will provide, we first discuss the general notion of monitor and monitoring system. We then identify, already in this abstract setting, the various requirements that give rise to the hierarchy of monitorability, depicted in the middle part of Fig. 1.1. Sec. 4 will then provide operational semantics to this hierarchy, in the setting of regular properties.

3.1 Monitoring Sytems

It is important to agree up-front on what properties are common to any reasonable monitoring framework. 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, Def. 3.1.1: verdicts are thus given for *incomplete* traces. Secondly, verdicts must be *irrevocable*, Def. 3.1.2. These postulates make explicit two features shared by most monitorability definitions in the literature.

Definition 3.1 A *monitoring system* consists of a triple $\langle M, \mathbf{acc}, \mathbf{rej} \rangle$, where M is a nonempty set of monitors, $\mathbf{acc}, \mathbf{rej} \subseteq M \times \textsc{Act}^{\infty}$, and for every $m \in M$:

1. For every finfinite trace $f \in \textsc{Act}^{\infty}$:
   - $\mathbf{acc}(m, f)$ implies $\exists s \in \textsc{Act}^* \cdot \big(s \preceq f$ and $\mathbf{acc}(m, s)\big)$ and
   - $\mathbf{rej}(m, f)$ implies $\exists s \in \textsc{Act}^* \cdot \big(s \preceq f$ and $\mathbf{rej}(m, s)\big)$;
2. For every finite trace $s \in \textsc{Act}^*$:
   - $\mathbf{acc}(m, s)$ implies $\forall f \in \textsc{Act}^{\infty} \cdot \mathbf{acc}(m, sf)$ and
   - $\mathbf{rej}(m, s)$ implies $\forall f \in \textsc{Act}^{\infty} \cdot \mathbf{rej}(m, sf)$.

■

Remark 3.1 Finite automata do not satisfy the requirements of a monitoring system as defined in Def. 3.1 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 3.2 A monitoring system $(M, \mathbf{acc}, \mathbf{rej})$ 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*) $\mathbf{acc}(m_P, f)$ iff trace $f \in \textsc{Act}^{\infty}$ has a prefix that positively determines P;
(*ii*) $\mathbf{rej}(m_P, f)$ iff trace $f \in \textsc{Act}^{\infty}$ has a prefix that negatively determines P. ■

In Sec. 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 a monitor $m \in M$ to monitor for a property P, it needs to satisfy some requirements. The most important such requirement is *soundness*.

Definition 3.3 (Soundness) Monitor m is *sound* for property P if for all f:

- $\mathbf{acc}(m, f)$ implies $f \in P$, and
- $\mathbf{rej}(m, f)$ implies $f \notin P$. ■

Remark 3.2 The definition of a monitoring system, Def. 3.1, does not preclude inconsistent monitors *i.e.*, there could be an $m \in M$ and an $f \in \textsc{Act}^{\infty}$ such that $\mathbf{acc}(m, f)$ and $\mathbf{rej}(m, f)$. Soundness for a property P does however prohibit inconsistences; in the instance above, it would imply both $f \in P$ and $f \notin P$, which is not possible. ■

Lemma 3.1 *If monitor m is sound for property P then*

- *if* $\boldsymbol{acc}(m, s)$*, then s positively determines P, and*
- *if* $\boldsymbol{rej}(m, s)$*, then s negatively determines P.*

Proof Fix a $s \in \text{Act}^*$ where $\mathbf{acc}(m, s)$ and pick some $f \in \text{Act}^\infty$. By Def. 3.1.2 and $\mathbf{acc}(m, s)$ we know that $\mathbf{acc}(m, sf)$ and by soudness we obtain $sf \in P$. □

Lemma 3.2 *For every property $P \subseteq \text{Act}^\infty$ and monitor m_P in a maximal monitoring system* $(M, \boldsymbol{acc}, \boldsymbol{rej})$*:*

1. *m_P is sound for P; and*
2. *if m is a sound monitor for P then*
   - $\boldsymbol{acc}(m, f)$ *implies* $\boldsymbol{acc}(m_P, f)$
   - $\boldsymbol{rej}(m, f)$ *implies* $\boldsymbol{rej}(m_P, f)$.

Proof The first item is immediate from Defs. 3.2 and 3.3. For the second clause, pick a finfinite trace f such that $\mathbf{acc}(m, f)$. By Def. 3.1 we know there exists some finite $s \preceq f$ such that $\mathbf{acc}(m, s)$ and by Lem. 3.1 we know that s positively determines P. By Def. 3.2 we deduce that $\mathbf{acc}(m_P, s)$ and by $s \preceq f$ and Def. 3.1 we obtain $\mathbf{acc}(m_P, f)$. The case for $\mathbf{rej}(m, f)$ is analogous. □

3.2 Shades of completeness

We are now ready to define monitorability in terms of the guarantees that the monitors are expected to give. Soundness is not negotiable. The dual requirement to soundness, *i.e.*, *completeness*, entails that the monitor detects *all* violating and satisfying traces.

Definition 3.4 (Completeness) Monitor m is *satisfaction-complete* for P if $f \in P$ implies $\mathbf{acc}(m, f)$ and *violation-complete* for P if $f \notin P$ implies $\mathbf{rej}(m, f)$. 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. ■

However, as shown now in Prop. 3.1, completeness is only possible for trivial properties in the finfinite domain; in the infinite domain more properties *are* completely monitorable—see Sec. 8.

Proposition 3.1 *If m is sound and complete for P then $P = \text{Act}^\infty$ or $P = \emptyset$.*

Proof If $\varepsilon \in P$, then $\mathbf{acc}(m, \varepsilon)$, so from Def. 3.1, $\forall f \in \text{Act}^\infty.\ \mathbf{acc}(m, f)$. Due to the soundness of m, $P = \text{Act}^\infty$. Similarly, $P = \emptyset$ when $\varepsilon \notin P$. □

Given the consequences of requiring completeness, as evidenced by Prop. 3.1, we also consider weaker forms of completeness. The weaker the completeness guarantee, the more properties can be monitored.

Definition 3.5 (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* 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. ■

The following lemma explicits the relation between the monitorability classes of Def. 3.5 and finite prefixes that determine a property, which are also called good and bad prefixes [41].

Lemma 3.3 *If $P \subseteq \textsc{Act}^\infty$ is monitorable for satisfaction (resp., for violation) by any monitoring system, then every $f \in P$ (resp., $f \in \textsc{Act}^\infty \setminus P$) has a finite prefix that positively (resp., negatively) determines P.*

Proof We treat the case for satisfaction, as the case for violation is dual. Let $f \in P$ and m be a monitor that is sound and satisfaction-complete for P. Then, due to satisfaction-completeness, $\mathbf{acc}(m, f)$, and by the requirements of Def. 3.1, there is a finite prefix s of f, such that $\mathbf{acc}(m, s)$. Therefore, by the same requirements, for every $g \in \textsc{Act}^\infty$, $\mathbf{acc}(m, sg)$. As we know that m is sound for P, this yields that s positively determines P. □

Since even partial monitorability, the weakest form in Def. 3.5, renders a substantial number of properties unmonitorable [5], 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 observed prefix the monitor is still able to reach a verdict (possibly after observing more actions).

Definition 3.6 (Informative Monitors[3]) A monitor m is:

- informatively accepting if there is trace that m accepts: $\exists f \in \textsc{Act}^\infty \cdot \mathbf{acc}(m, f)$;
- informatively rejecting if there is a trace that m rejects: $\exists f \in \textsc{Act}^\infty \cdot \mathbf{rej}(m, f)$;
- informative when it either accepts or rejects a trace:
  $\exists f \in \textsc{Act}^\infty \cdot \mathbf{rej}(m, f)$ or $\mathbf{acc}(m, f)$;
- persistently accepting if it remains informatively accepting for all finite traces:
  $\forall s \in \textsc{Act}^* \cdot \exists f \cdot \mathbf{acc}(m, sf)$;
- persistently rejecting if it remains informatively rejecting for all finite traces:
  $\forall s \in \textsc{Act}^* \cdot \exists f \cdot \mathbf{rej}(m, sf)$;
- persistently informative when it remains informative for all finite traces:
  $\forall s \in \textsc{Act}^* \cdot \exists f \cdot \mathbf{rej}(m, sf)$ or $\mathbf{acc}(m, sf)$. ■

Definition 3.7 (Informative Monitorability) We say that:

- A property P is informatively monitorable for satisfaction (*resp.,* for violation) if there is an informatively accepting (*resp.,* informatively rejecting) monitor that is sound for P.

[3] These are *not* related to the *informative prefixes* from [41] or to *persistence* from [49].

– A property P is informatively monitorable if there is a informative monitor that is sound for P.
– A property P is persistently informatively monitorable for satisfaction (*resp.,* for violation) if there is a persistently accepting (*resp.,* persistently rejecting) monitor that is sound for P.
– A property P is persistently informatively monitorable if there is a persistently informative monitor that is sound for P.
– A class of properties $C \subseteq 2^{\text{ACT}^\infty}$ 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 (*resp.,* PICmp). ■

Example 3.1 Recall the property *"f never occurs and eventually s is reached"* from Example 1.1 (expressible in LTL as $(\mathsf{G}\,\neg\mathsf{f}) \wedge (\mathsf{F}\,\mathsf{s})$). Given any maximal monitoring system, this property is *not* partially monitorable: a monitor cannot accept the satisfying infinite trace $\mathsf{s}(\mathsf{r})^\omega$ by just observing a finite prefix, nor can it reject the violating trace r^ω by observing one of its finite prefixes. It is, however, persistently informatively monitorable for violation: every finite prefix that is not yet violating can be extended to produce the action f which would be enough evidence for a monitor to reject the trace. ■

Example 3.2 The property requiring that *"r only appears a finite number of times"* is *not* informatively monitorable. If it were, the respective sound informative monitor m in the maximal system should at least accept or reject one trace. If it accepts a trace f, by Def. 3.1, it must accept some prefix $s \preceq f$. Again, by Def. 3.1, all continuations, including sr^ω, must be accepted by m. This makes it unsound, which is a contradiction. A dual argument can also be made for rejections. 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 3.1 (Monitorability Hierarchy) *In any maximal monitoring system, the monitorability classes given in Defs. 3.5 and 3.7 form the inclusion hierarchy depicted in Fig. 1.1(middle).*

Proof The only non-trivial inclusion to show from Fig. 1.1(middle) is

$$\mathsf{PCmp} = \mathsf{SCmp} \cup \mathsf{VCmp} \subseteq \mathsf{PICmp}.$$

Pick a property $P \in \mathsf{VCmp}$. Pick also a finite trace $s \in \text{ACT}^*$. If $sf \notin P$ for some f, then by Def. 3.4 we have $\mathbf{rej}(m_P, sf)$. Otherwise, $sf \in P$ for each f, meaning that s positively determines P, and by Def. 3.2 we have $\mathbf{acc}(m_P, sf)$. By Def. 3.6, 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 Def. 3.7, it follows that $P \in \mathsf{PICmp}$. The case for $P \in \mathsf{SCmp}$ is dual. □

Remark 3.3 We note that a property being partially monitorable does *not* imply that it is also persistently informatively monitorable for satisfaction or for violation. Furthermore, not all persistently informatively monitorable properties are also informatively monitorable for satisfaction, and they are not all informatively monitorable for violation. To see why this is the case, simply observe that tt is not informatively monitorable for violation. ■

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

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

Fig. 4.1 recHML Syntax and (finfinite) Linear-Time Semantics

4 An Instantiation for Regular Properties

We now 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 the next section.

We use the logic recHML to represent regular properties. This is a reformulation of the modal μ-calculus [40], and embeds other specification formalisms such as LTL, (ω-)regular expressions, Büchi automata, and Street automata, used in the state of the art on monitorability.

We begin by recalling the syntax and semantics of recHML and the monitoring system for regular properties from [5]. We then argue that this monitoring system is maximal for regular properties, in the sense of Def. 3.2, and show that this means that it subsumes all other monitoring systems for regular properties. This both demonstrates that the framework proposed in Sec. 3 is realistic and allows us to work with a fixed monitoring system in the sequel without loss of generality.

4.1 The Logic.

The syntax of recHML is defined by the grammar in Fig. 4.1, 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. 4.1. It maps a (possibly open) formula to a set of (finfinite) traces [5] by induction on the formula structure, using valuations that map logical variables to sets of traces, $\sigma : \textsc{LVar} \to \mathcal{P}(\textsc{Act}^{\infty})$, 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 least and greatest fixpoint formulae, say $\mathsf{min}\,X.\varphi$ 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 [42]. In the discussions we occasionally treat formulae, φ, as the properties they denote, $[\![\varphi]\!]$.

LTL [23] is the specification logic of choice for many RV approaches. As a consequence, it is also the logic used by a number of studies on monitorability (*e.g.,* see [16, 17, 37]). Our choice of logic, recHML, is not limiting in this regard because it is well known [40, 54] that LTL can be translated into recHML.

Example 4.1 The characteristic LTL operators can be encoded in recHML as:

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

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

The use of recHML allows us to consider monitorable properties that may be missed 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 [54].

Example 4.2 Recall the system discussed in Example 1.1 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 [54], it can be expressed in recHML as:

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

Note that LTL properties such as $\neg\mathsf{s} \wedge \mathsf{G}(\mathsf{s} \Leftrightarrow \neg\mathsf{s})$ do not express the aforementioned property; the LTL property given is in fact too strict (it describes "s at even positions only") and rules out traces of the form f^ω which clearly satisfy the property φ_{even}. 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:

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

More broadly, recHML captures all (ω-)regular properties, while LTL can only express properties recognised by counter-free Büchi automata [28]. Our logic of choice has several other advantages over LTL:

- recHML semantics adapt easily to the finite, infinite and finfinite domains. LTL semantics are only standard on infinite traces; there is no canonical finite or finfinite semantics. (See, however, [26] for a finite-trace semantics for LTL and Linear Dynamic Logic.). In particular, to specify whether a property holds or not on a finite trace, we would need to add to the syntax of LTL modalities corresponding to the box and diamond of recHML that indicate whether a continuation is allowed or required, thus moving away from standard LTL.
- recHML is closer to the underlying automata models, and to the process algebras describing our monitors. For instance, given a monitor, it is straightforward to deduce the recHML formula for which it is sound and complete; however, it is nontrivial to even decide whether such an LTL formula exists.

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

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

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

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

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

Fig. 4.2 Monitor Syntax and Labelled-Transition Semantics

Therefore, to study monitorability, we prefer to use recHML, as it can express all regular properties, allowing for clearer distinctions between monitorability classes. Furthermore, when synthesizing a monitor, one can use a specification in LTL, embed it to recHML in a straightforward manner, and then use a monitor synthesis that relies on recHML, thus gaining all advantages these logics offer. For the sake of better readability, and in the light of its familiarity to the RV community, we use LTL for the examples that can be encoded in that logic. Note that, since we operate in the finfinite domain, X should be read as a *strong* next operator, in line with Example 4.1.

In the sequel, we use the following classical result for recHML$_{\text{B}}$-like specification logics (see [10] for more on the μ-calculus and recHML):

Lemma 4.1 *If $\varphi \in$ recHML, then $[\![\varphi]\!] \cap \textsc{Act}^*$ is regular.*

Lemma 4.2 *If $\varphi \in$ recHML, then D^+_φ and D^-_φ are regular.*

Proof We know that $[\![\varphi]\!] \cap \textsc{Act}^*$ is regular (Lem. 4.1) and $[\![\varphi]\!] \cap \textsc{Act}^\omega$, the infinite-trace interpretation of φ, is ω-regular. Therefore, there are a DFA D_F that recognizes $[\![\varphi]\!] \cap \textsc{Act}^*$ and a deterministic ω-automaton D_I that recognizes $[\![\varphi]\!] \cap \textsc{Act}^\omega$. Let $A_F = \{s \in \textsc{Act}^* \mid \forall r \in \textsc{Act}^*.\ sr \in [\![\varphi]\!]\}$ and $A_I = \{s \in \textsc{Act}^* \mid \forall t \in \textsc{Act}^\omega.\ st \in [\![\varphi]\!]\}$. Let Q_F (*resp.*, Q_I) be the set of states in D_F (*resp.*, in D_I) that can be reached reading some trace $s \in A_F$(*resp.*,A_I). By construction, for each $s \in \textsc{Act}^*$, we have that $s \in A_F$ (*resp.*, $s \in A_I$) if and only if s does not end in Q_F (*resp.*, Q_I). Therefore, there are DFAs D'_F and D'_I for A_F and A_I, respectively, and thus $D^+_\varphi = A_F \cap A_I$ is regular. The case for D^-_φ is similar. □

4.2 The Monitors.

We consider the operational monitoring system of [5, 35], summarised in Fig. 4.2 (symmetric rules for binary operators are omitted). 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. The syntax in Fig. 4.2 assumes a countably infinite set of variables $x, y, \ldots \in \textsc{Vars}$; see [5] for a comprehensive discussion.

The monitoring system $(\textsc{Mon}, \mathbf{acc}, \mathbf{rej})$ is given by a *labelled transition system (LTS)* based on $\textsc{Act}$, which is comprised of the monitor states, or monitors, and a transition relation. The set of monitor states, $\textsc{Mon}$, and the monitor transition relation, $\longrightarrow \subseteq (\textsc{Mon} \times (\textsc{Act} \cup \{\tau\}) \times \textsc{Mon})$, are defined in Fig. 4.2. The suggestive notation $m \xrightarrow{\mu} n$ denotes $(m, \mu, n) \in \longrightarrow$; we also write $m \not\xrightarrow{\mu}$ to denote $\neg(\exists n.\ m \xrightarrow{\mu} n)$. We employ the usual notation for weak transitions and write $m \Longrightarrow n$ in lieu of $m(\xrightarrow{\tau})^* n$ and $m \xRightarrow{\mu} n$ for $m \Longrightarrow \cdot \xrightarrow{\mu} \cdot \Longrightarrow n$, where $\mu \in (\textsc{Act} \cup \{\tau\})$. We write sequences of transitions $m \xRightarrow{a_1} \cdots \xRightarrow{a_k} n$ as $m \xRightarrow{s} n$, where $s = a_1 \cdots a_k$. A monitor that does not use any parallel operator is called a *regular monitor*. The full monitoring system and regular monitors were defined and used in [1, 4, 5, 32, 33, 35]. We refer the interested reader to these studies for explanations and motivations.

This semantics gives an operational account of how a monitor in state m incrementally analyses a sequence of actions $s = a_1 \ldots a_k$ to reach a new monitor state n; the monitor m accepts (*resp.*, rejects) a trace f, written $\mathbf{acc}(m, f)$ (*resp.*, $\mathbf{rej}(m, f)$), when it can transition to the verdict state yes (*resp.*, no) while analysing a prefix $s \preceq f$.

Definition 4.1 (Acceptance and Rejection) For a monitor $m \in \textsc{Mon}$, we define $\mathbf{rej}(m, s)$ (*resp.*, $\mathbf{acc}(m, s)$) and say that m *rejects* (*resp.*, *accepts*) when $m \xRightarrow{s} \mathsf{no}$ (*resp.*, $m \xRightarrow{s} \mathsf{yes}$). Similarly, for $t \in \textsc{Act}^\omega$, we write $\mathbf{rej}(m, t)$ (*resp.*, $\mathbf{acc}(m, t)$) if there exist $s \in \textsc{Act}^*$ and $u \in \textsc{Act}^\omega$ such that $t = su$ and m rejects (*resp.*, accepts) s. ■

For a finite nonempty set of indices I, we use $\sum_{i \in I} m_i$ to denote any combination of the monitors in $\{m_i \mid i \in I\}$ using the operator $+$. For each $j \in I$, $\sum_{i \in I} m_i$ is called a sum of m_j, and m_j is called a summand of $\sum_{i \in I} m_i$. The following Lem. 4.3 assures us that regular monitors satisfy the conditions to be a monitoring system, given in Def. 3.1.

Lemma 4.3 (Verdict Persistence, [5, 35]) *$v \xRightarrow{s} m$ implies $m = v$.*

We will use the following definitions and results in our proofs. We define determinism for regular monitors.

Definition 4.2 ([3, 4]) A closed regular monitor m is *deterministic* iff every sum of at least two summands that appears in m is of the form $\sum_{\alpha \in A} \alpha.m_\alpha$, where $A \subseteq \textsc{Act}$. ■

Definition 4.3 (Verdict Equivalence) Monitors m and n are called *verdict equivalent* when for every $f \in \textsc{Act}^\infty$, $\mathbf{acc}(m, f)$ iff $\mathbf{acc}(n, f)$ and $\mathbf{rej}(m, f)$ iff $\mathbf{rej}(n, f)$. ■

Theorem 4.1 ([4, 5]) *Every monitor in* $\textsc{Mon}$ *is verdict equivalent to a deterministic regular monitor.*

Due to Thm. 4.1, we can assume that every monitor in $\textsc{Mon}$ is a regular, or deterministic regular monitor. We often do so in the following proofs.

Theorem 4.2 **([3,4])** *If $L, L' \subseteq \textsc{Act}^*$ are regular and suffix-closed, and $L \cap L' = \emptyset$, then there is a regular monitor m, such that $\textbf{acc}(m,s)$ iff $s \in L$ and $\textbf{rej}(m,s)$ iff $s \in L'$.*

We use the formula synthesis function from regular monitors to formulae defined in [5,35] (we assume a bijection between logical and monitor variables that we leave implicit):

$$\begin{array}{ccc} \mathsf{f}(\mathsf{no}) = \mathsf{ff} & \mathsf{f}(\mathsf{end}) = \mathsf{f}(\mathsf{yes}) = \mathsf{tt} & \mathsf{f}(x) = X \\ \mathsf{f}(m+n) = f(m) \wedge \mathsf{f}(n) & \mathsf{f}(a.m) = [a]\mathsf{f}(m) & \mathsf{f}(\mathsf{rec}\, X.m) = \mathsf{max}\, X.\mathsf{f}(m) \end{array}$$

Theorem 4.3 **([5])** *Every regular monitor m is sound and violation-complete for $f(m)$.*

From properties such as Lem. 4.3 is not hard to see that this operational framework satisfies the conditions for a monitoring system of Def. 3.1. The monitoring system of Fig. 4.2 is also maximal for regular properties, according to Def. 3.2. This concrete instance thus demonstrates the realisability of the abstract definitions in Sec. 3.

Theorem 4.4 *For all $\varphi \in \textsc{rec}$HML, there is a regular monitor m that is sound for φ and accepts all finite traces that positively determine φ and rejects all finite traces that negatively determine φ.*

Proof By Lem. 4.2, D^+_φ and D^-_φ, the sets of finite traces that (respectively) positively or negatively determine φ are regular. It is also not hard to see that they are suffix-closed. Therefore the theorem follows from Thm. 4.2. □

As a corollary of Thm. 4.4, from Lem. 3.1 we deduce that for any arbitrary monitoring system $(M, \textbf{acc}, \textbf{rej})$, if $m \in M$ is sound for some $\varphi \in \textsc{rec}$HML, then there is a monitor $n \in \textsc{Mon}$ from Fig. 4.2 that accepts (*resp.*, rejects) all traces f that m accepts (*resp.*, rejects).

Corollary 4.1 *If m is a sound monitor for $\varphi \in \textsc{rec}$HML, then there is a regular monitor n that is sound for φ, and such that for every $s \in \textsc{Act}^*$, $\textbf{acc}(m,s)$ implies $\textbf{acc}(n,s)$, and $\textbf{rej}(m,s)$ implies $\textbf{rej}(n,s)$.*

Proof By Thm. 4.4, there is a regular monitor n that is sound for φ, and accepts all finite traces that positively determine φ, and rejects all the finite traces that negatively determine φ. If $\textbf{acc}(m,s)$ (*resp.*, $\textbf{rej}(m,s)$) for some finite trace s, then, due to the soundness of m, $s \in [\![\varphi]\!]$ (*resp.*, $s \notin [\![\varphi]\!]$), and therefore, from Lem. 3.1, s positively (*resp.*, negatively) determines φ. By the properties of n, we have that $\textbf{acc}(n,s)$ (*resp.*, $\textbf{rej}(n,s)$). □

In the sequel, we thus assume $(\textsc{Mon}, \textbf{acc}, \textbf{rej})$ from Fig. 4.2 as our fixed monitoring system, as it subsumes all others.

5 A Syntactic Characterisation of Monitorability

We present syntactic characterisations for the various monitorability classes as fragments of $\textsc{rec}$HML. We begin by recalling the syntactic characterisation of partial monitorability by Aceto *et al.* from [5], and then proceed to provide the corresponding syntactic characterisations for informative and persistently informative monitorability. The fragments we provide are *maximal* in the sense that they not only guarantee that any property expressible within the fragment is monitorable with the corresponding guarantees, but also conversely, every property that is monitorable with respect to the corresponding notion of monitorability is expressible in the fragment.

5.1 Partial Monitorability, syntactically.

In [5], Aceto *et al.* identify a maximal partially monitorable syntactic fragment of recHML.

Theorem 5.1 (Partially-Complete Monitorability [5]) *Consider the syntactic fragments:*

$$\varphi,\psi \in \text{sHML} ::= \mathsf{tt} \mid \mathsf{ff} \mid [a]\varphi \mid \varphi\wedge\psi \mid \mathsf{max}\, X.\varphi \mid X \textit{ and}$$
$$\varphi,\psi \in \text{cHML} ::= \mathsf{tt} \mid \mathsf{ff} \mid \langle a\rangle\varphi \mid \varphi\vee\psi \mid \mathsf{min}\, X.\varphi \mid X.$$

The fragment sHML *is monitorable for violation whereas* cHML *is monitorable for satisfaction. Furthermore, if* $\varphi \in$ recHML *is monitorable for satisfaction (resp., for violation) by some* $m \in$ Mon*, it is expressible in* cHML*(resp.,* sHML*), i.e.,* $\exists\psi\in$cHML *(resp.,* $\psi\in$sHML*), such that* $[\![\varphi]\!]=[\![\psi]\!]$.

Observe that for every regular monitor m, $\mathsf{f}(m) \in$ sHML.

As a corollary of Thm. 5.1 we obtain *maximality*: any $\varphi \in$ recHML that is monitorable for satisfaction (*resp.,* for violation) can also be expressed as some $\psi \in$ cHML (*resp.,* $\psi \in$ sHML) where $[\![\varphi]\!] = [\![\psi]\!]$. For this fragment, the following automated synthesis function, which is readily implementable, is given in [5].

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

5.2 Informative monitorability, syntactically.

We proceed to identify syntactic fragments of recHML that correspond to informative monitorability. Intuitively, a sHML formula is informatively monitorable for violation if ff appears in it: there is a trace that falsifies the formula. Furthermore, the conjunction of any such formula with an arbitrary formula is still falsified by the same trace. Dually, cHML formulas in which tt occurs are informatively monitorable for satisfaction, and so are their disjunctions with arbitrary formulas. We now formalise this intuition.

Definition 5.1 The informative fragment is iHML = siHML ∪ ciHML where

$$\text{siHML} = \{\varphi_1 \wedge \varphi_2 \in \text{recHML} \mid \varphi_1 \in \text{sHML and } \mathsf{ff} \text{ appears in } \varphi_1\},$$
$$\text{ciHML} = \{\varphi_1 \vee \varphi_2 \in \text{recHML} \mid \varphi_1 \in \text{cHML and } \mathsf{tt} \text{ appears in } \varphi_1\}. \quad \blacksquare$$

We define the depth of ff in an sHML formula in a recursive way: $d_{\mathsf{ff}}(\mathsf{ff}) = 0$; $d_{\mathsf{ff}}(\mathsf{tt}) = d_{\mathsf{ff}}(X) = \infty$; $d_{\mathsf{ff}}(\psi_1 \wedge \psi_2) = \min\{d_{\mathsf{ff}}(\psi_1), d_{\mathsf{ff}}(\psi_2)\} + 1$; $d_{\mathsf{ff}}([\alpha]\psi) = d_{\mathsf{ff}}(\psi) + 1$; and $d_{\mathsf{ff}}(\max X.\psi) = d_{\mathsf{ff}}(\psi) + 1$.

Lemma 5.1 *For all possibly open $\varphi, \psi \in$ sHML $d_{\mathsf{ff}}(\varphi[\psi/X]) \leq d_{\mathsf{ff}}(\varphi)$.*

Proof Straightforward induction on φ. □

Lemma 5.2 *If $\varphi \in$ siHML (resp., ciHML), then there is a regular monitor that is sound and informatively rejecting (resp., informatively accepting) for φ. If $\varphi \in$ iHML, then there is a regular monitor that is sound and informative for φ.*

Proof We assume that $\varphi \in$ siHML, as the case for $\varphi \in$ ciHML is similar. Let $\varphi = \varphi_1 \wedge \varphi_2$, where $\varphi_1 \in$ sHML and ff appears in φ_1. First of all, we prove by strong numerical induction on $d_{\mathsf{ff}}(\psi)$ that for all $\psi \in$ sHML, if $d_{\mathsf{ff}}(\psi) < \infty$, then there is a finite trace that negatively determines ψ. If $d_{\mathsf{ff}}(\psi) = 0$, then $\psi = \mathsf{ff}$, and we are done, as ε negatively determines ff. Otherwise, $d_{\mathsf{ff}}(\psi) = k + 1$ and we consider the following cases:

$\psi = \psi_1 \wedge \psi_2$ In this case, either $d_{\mathsf{ff}}(\psi_1) = k$ or $d_{\mathsf{ff}}(\psi_2) = k$, so by the inductive hypothesis, there is a finite trace that negatively determines one of the two conjuncts, and therefore also ψ.

$\psi = [\alpha]\psi'$ In this case, $d_{\mathsf{ff}}(\psi') = k$, so, by the inductive hypothesis, there is a finite trace s that negatively determines ψ', so αs negatively determines ψ.

$\psi = \max X.\psi'$ In this case, $d_{\mathsf{ff}}(\psi') = k$. Therefore, from Lem. 5.1, $d_{\mathsf{ff}}(\psi'[\psi/X]) \leq d_{\mathsf{ff}}(\psi') = k$, so, by the inductive hypothesis, there is a finite trace s that negatively determines $\psi'[\psi/X]$, so it also negatively determines ψ, because $[\![\psi'[\psi/X]]\!] = [\![\psi]\!]$.

As ff appears in φ_1, $d_{\mathsf{ff}}(\varphi) < \infty$, so there is a finite trace that negatively determines φ_1, and therefore also φ. The lemma follows from Thm. 4.4. □

Lemma 5.3 *If $\varphi \in$ recHML and there is a monitor that is sound and informatively accepting (resp., informatively rejecting) for φ, then there is some $\psi \in$ ciHML (resp., siHML) such that $[\![\psi]\!] = [\![\varphi]\!]$.*

Proof If m is sound and informatively accepting for φ, then by Lem. 3.1, there is a finite trace s that positively determines φ. We can then easily construct a formula $\psi_1(s)$ that is satisfied exactly by s and all its extensions, recursively on s: let $\psi_1(\varepsilon) = \mathsf{tt}$, and let $\psi_1(\alpha s) = \langle\alpha\rangle\psi_1(s)$. Then, let $\psi = \psi_1(s) \vee \varphi$. Thus, $\psi \in$ ciHML and $[\![\psi]\!] = [\![\varphi]\!]$. The case for informatively rejecting monitors is similar. □

Theorem 5.2 *For $\varphi \in$ recHML, φ is informatively monitorable for violation (resp., satisfaction) if and only if there is some $\psi \in$ siHML (resp., ciHML) such that $[\![\psi]\!] = [\![\varphi]\!]$. φ is informatively monitorable if and only if there is some $\psi \in$ iHML, such that $[\![\psi]\!] = [\![\varphi]\!]$.*

Proof A consequence of Lems. 5.2 and 5.3. □

The maximality results of Thms. 5.1 and 5.2 permit tool constructions to concentrate on the syntactic fragments identified when synthesizing monitors. To achieve the corresponding monitorability guarantees, one would have to first work on the given formula and find an appropriate equivalent form in the right fragment. Thms. 5.1 and 5.2 also serve as a syntactic check to determine when a property is monitorable (according to the monitorability classes in Fig. 1.1). We note that these syntactic characterisations may not always yield monitors that detect as many satisfactions or violations as they could. However, Thm. 4.4 assures us that for each recHML formula φ, there is a monitor m that detects all traces that positively or negatively determine φ, and therefore that monitor will satisfy all guarantees that are possible when monitoring φ, and therefore the knowledge that φ is in a certain fragment informs us of a certain good behaviour of m.

Example 5.1 The property φ_{evenW} from Example 4.2 is monitorable for violation; this can be easily determined since it is expressible in sHML. By contrast, φ_{even} from Example 4.2 cannot be expressed in either sHML or cHML. In fact, it is *not* partially-complete monitorable: it cannot be monitored completely for satisfaction because the trace $(\mathsf{rs})^\omega \in [\![\varphi_{\text{even}}]\!]$ but none of its prefixes can be accepted by a sound monitor since they all violate the property; it cannot be monitored completely for violation 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 3.1 (expressed here in LTL) is a siHML property, as $\mathsf{G}\neg\mathsf{f}$ can be written in sHML as $\mathsf{max}\,X.[\mathsf{f}]\mathsf{ff}\wedge[\mathsf{s}]X\wedge[\mathsf{r}]X$. In contrast, $\mathsf{FG}\neg\mathsf{r}$ cannot be written in iHML since it is not informatively monitorable. ■

Remark 5.1 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. One obvious choice is a formula describing D^+, the set of finite traces that positively determines a property, and dually the formula describing $\text{Act}^\infty \setminus D^-$, the set of traces that do not negatively determine the property. See Kupferman and Vardi's construction in [41] for how to construct these formulae; this method has to be adapted a little in the finfinite domain, but this is outside the scope of the present work. ■

5.3 Persistently informative monitorability for satisfaction and violation, syntactically.

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.

We begin by characterising persistently informative monitorability for satisfaction and for violation separately. The following definition of explicit formulae forces modal subformulae to explicitly list every action. Observe that a disjunction of existential modalities requires there to be a successor while the conjunction of universal modalities holds if there is no successor.

Definition 5.2 We define eHML, the explicit fragment of recHML:

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

Example 5.2 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 5.3 Let φ be a closed recHML formula and let ψ be a subformula of φ. We say that:

- ψ can refute (*resp.*, verify) in φ in 0 unfoldings, when ff (*resp.*, tt) appears in ψ, and that
- ψ can refute (*resp.*, verify) in φ 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 $\mathsf{max}\, X.\psi'$ or $\mathsf{min}\, X.\psi'$ that can refute (*resp.*, verify) in k unfoldings.

We simply say that ψ can refute (*resp.*, verify) in φ when it can refute (*resp.*, verify) in φ in k unfoldings, for some $k \geq 0$. We may also simply say that ψ can refute (*resp.*, verify) when φ is evident or not relevant. ■

Example 5.3 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 can define a similar notion for monitors.

Definition 5.4 Let m be a closed monitor and let n be a submonitor of m. We say that:

- n can reject (*resp.*, accept) in m in 0 unfoldings, when no (*resp.*, yes) appears in n, and that
- n can reject (*resp.*, accept) in m in $k+1$ unfoldings, when it can reject (*resp.*, accept) in k unfoldings, or x appears in n and n is in the scope of a submonitor $\mathsf{rec}\, X.n'$ that can reject (*resp.*, accept) in k unfoldings.

We simply say that n can reject (*resp.*, accept) in m when it can reject (*resp.*, accept) in m in k unfoldings, for some $k \geq 0$. We may also simply say that n can reject (*resp.*, accept) when m is evident or not relevant. ■

We now define the fragments of recHML corresponding to recHML properties that are persistently informatively monitorable for satisfaction or violation. The intuition is similar to the one underlying the definition of the informative fragment, except here the reachability condition is quantified universally over subformulae, and we need the informative part of the formula to be explicit.

Definition 5.5 We define the fragments sᴘHML and cᴘHML as:

$$\textsc{sp}\mathrm{HML} = \left\{ \varphi_1 \wedge \varphi_2 \in \textsc{rec}\mathrm{HML} \;\middle|\; \begin{array}{l} \varphi_1 \in \mathrm{sHML} \cap \mathrm{eHML} \text{ and every} \\ \text{subformula of } \varphi_1 \text{ can refute} \end{array} \right\}$$

$$\textsc{cp}\mathrm{HML} = \left\{ \varphi_1 \vee \varphi_2 \in \textsc{rec}\mathrm{HML} \;\middle|\; \begin{array}{l} \varphi_1 \in \mathrm{cHML} \cap \mathrm{eHML} \text{ and every} \\ \text{subformula of } \varphi_1 \text{ can verify} \end{array} \right\} \qquad \blacksquare$$

We now make explicit two (obvious) lemmas used in the sequel.

Lemma 5.4 *Let $\varphi = \mathsf{max}\, X.\psi$ or $\varphi = \mathsf{min}\, X.\psi$. If φ can refute (resp., verify) in φ, then it is also the case that $\psi[\varphi/X]$ can refute (resp., verify) in $\psi[\varphi/X]$.*

Lemma 5.5 – *If all subformulae of $[\alpha]\varphi$ or $\varphi \wedge \psi$ or $\psi \wedge \varphi$ or $\langle\alpha\rangle\varphi$ or $\varphi \vee \psi$ or $\psi \vee \varphi$ can refute (or, respectively, verify), then all subformulae of φ can refute (or verify).*
– *Let $\varphi = \mathsf{max}\, X.\psi$ or $\varphi = \mathsf{min}\, X.\psi$. If all subformulae of φ can refute (resp., verify), then all subformulae of $\psi[\varphi/X]$ can refute (resp., verify).*

We define the box-depth of a formula from eHML ∩ sHML recursively:

$$\begin{aligned} d_B\left(\bigwedge_{\gamma\in\textsc{Act}} [\gamma]\varphi_\gamma\right) &= d_B(\mathsf{ff}) = 0; \\ d_B(X) &= d_B(\mathsf{tt}) = \infty; \\ d_B(\varphi_1 \wedge \varphi_2) &= \min\{d_B(\varphi_1), d_B(\varphi_2)\} + 1; \qquad \text{and} \\ d_B(\max X.\varphi') &= d_B(\varphi') + 1. \end{aligned}$$

The box-depth of a formula measures how deep in the syntactic tree of the formula one can find a box or ff.

Lemma 5.6 *For all possibly open $\varphi, \psi \in \mathrm{eHML} \cap \mathrm{sHML}$ $d_B(\varphi[\psi/X]) \leq d_B(\varphi)$.*

Proof Straightforward induction on φ. □

Lemma 5.7 *Let $\alpha \in \textsc{Act}$.*

– *Let $\varphi \in \mathrm{eHML} \cap \mathrm{sHML}$, where all subformulae of φ can refute. There is some $\psi \in \mathrm{eHML} \cap \mathrm{sHML}$, such that all subformulae of ψ can refute, and for every $f \in \textsc{Act}^\infty$, $\alpha f \in [\![\varphi]\!]$ implies that $f \in [\![\psi]\!]$.*
– *Let $\varphi \in \mathrm{eHML} \cap \mathrm{cHML}$, where all subformulae of φ can verify. There is some $\psi \in \mathrm{eHML} \cap \mathrm{cHML}$, such that all subformulae of ψ can verify, and for every $f \in \textsc{Act}^\infty$, $f \in [\![\psi]\!]$ implies that $\alpha f \in [\![\varphi]\!]$.*

Proof We assume that $\varphi \in \mathrm{eHML} \cap \mathrm{sHML}$, as the case for $\varphi \in \mathrm{eHML} \cap \mathrm{cHML}$ is similar. Since φ is a closed formula and can refute, ff appears in φ, and therefore $d_B(\varphi) < \infty$. We proceed to prove the lemma by strong numerical induction on $d_B(\varphi)$, similarly to the proof of Lem. 5.2.

If $\varphi = \mathsf{ff}$, then we are done immediately by taking $\psi = \mathsf{ff}$.
If $\varphi = \bigwedge_{\gamma\in\textsc{Act}} [\gamma]\varphi_\gamma$, then we can set $\psi = \varphi_\alpha$.

If $\varphi = \varphi_1 \wedge \varphi_2$, then either $d_a(\varphi_1) < \infty$ or $d_a(\varphi_2) < \infty$, and we are done by the inductive hypothesis on one of the two subformulae.

If $\varphi = \max X.\varphi'$, then $\varphi'[\varphi/X] \in$ eHML $\cap$ sHML and all subformulae of $\varphi'[\varphi/X]$ can refute, by Lem. 5.5. Furthermore, $[\![\varphi]\!] = [\![\varphi'[\varphi/X]]\!]$, and we are done by the inductive hypothesis. □

Lemma 5.8 *If $\varphi \in$ spHML or $\varphi \in$ cpHML, then there is a regular monitor that is sound for φ and persistently rejecting, or, respectively, persistently accepting.*

Proof We assume that $\varphi \in$ spHML, as the case for $\varphi \in$ cpHML is similar. Let $\varphi = \psi \wedge \psi_*$, where $\psi \in$ eHML $\cap$ sHML and all of its subformulae can refute, and $\psi_* \in$ recHML. By Thm. 4.4, it suffices to prove that for every $s \in$ Act*, there is some $r \in$ Act*, such that sr negatively determines φ. We prove this by structural induction on s. If $s = \varepsilon$, then as in the proof of Lem. 5.2, we can show that there is a finite trace that negatively determines ψ. If $s = as'$, then by Lem. 5.7. there is some $\psi' \in$ eHML$\cap$sHML, such that all subformulae of ψ' can refute, and for every $f \in$ Act$^\infty$, $af \in [\![\psi]\!]$ implies that $f \in [\![\psi']\!]$. By the inductive hypothesis, there is some r, such that $s'r$ negatively determines ψ', and therefore, sr negatively determines ψ. □

We define the depth of a variable x in a regular monitor m recursively:

$$\begin{aligned} d_x(x) &= 0; \\ d_x(y) &= d(\mathsf{no}) = d(\mathsf{yes}) = d(\mathsf{end}) = \infty, && \text{where } y \neq x; \\ d_x(m_1 + m_2) &= \min\{d_x(m_1), d_x(m_2)\} + 1; \\ d_x(\alpha.m) &= d_x(m) + 1; && \text{and} \\ d_x(\mathsf{rec}\, x.\ m) &= d_x(\mathsf{rec}\, y.\ m) = d_x(m) + 1. \end{aligned}$$

Lemma 5.9 *Let m be a persistently rejecting, deterministic regular monitor. If $A \subsetneq$ Act, then $\sum_{\alpha \in A} \alpha.m_\alpha$ can only appear in m as a submonitor of a larger sum.*

Proof Let $a \in$ Act$\backslash A$ and let m' be an open monitor and x a variable that does not appear in m, such that $m = m'[\sum_{\alpha \in A} \alpha.m_\alpha / x]$. It is clear that $\sum_{\alpha \in A} \alpha.m_\alpha \overset{a}{\not\Rightarrow}$. Therefore, it suffices to prove that for every deterministic n with free variable x, if $n' \overset{a}{\not\Rightarrow}$, then there is a finite trace s, such that there is no regular monitor o for which $n[n'/x] \overset{sa}{\Longrightarrow} o$. We proceed to prove this claim by induction on $d_x(n)$, and the case for $n = x$ is immediate. If $n = n_1 + n_2$, then, as n is deterministic, $n = b.n_1' + c.n_2'$, where $b \neq c$, and we are done by the inductive hypothesis on either n_1' or n_2', and n'. If $n = b.n_1$, then if the inductive hypothesis on n_1' and n' gives trace r, then we can set $s = br$. If $n = \mathsf{rec}\, y.n_1$, then we are done by the inductive hypothesis on $n_1[n/y]$ (notice that $d_x(n_1[n/y] < d_x(m))$ and $n'[n/y]$. □

Here we call a regular monitor explicit when it is generated by the grammar:

$$m ::= \mathsf{end} \quad | \quad \mathsf{no} \quad | \quad x \quad | \quad \sum_{\alpha \in Act} \alpha.m_\alpha \quad | \quad \mathsf{rec}\, x.m.$$

Corollary 5.1 *Every persistently rejecting, deterministic regular monitor is explicit.*

Proof From Lem. 5.9. □

Lemma 5.10 *Let m be an explicit deterministic regular monitor, such that all of its submonitors can reject. Then, $\mathsf{f}(m) \in$ EHML and all of its subformulae can refute.*

Proof By induction on the construction of m. □

Lemma 5.11 *If $\varphi \in$ RECHML and there is a monitor that is sound for φ and persistently rejecting or persistently accepting, then there is some $\psi \in$ SPHML, or, respectively, $\psi \in$ CPHML, such that $[\![\psi]\!] = [\![\varphi]\!]$.*

Proof We treat the case where the monitor is persistently rejecting, as the case for a persistently accepting monitor is similar. From Cor. 4.1, there is a regular monitor, m, that is sound for φ and persistently rejecting. By Thm. 4.1, we can assume that m is deterministic (Def. 4.2). From Cor. 5.1, m is explicit. If there is a submonitor of m that cannot reject, then we can prove by induction on m that there is a finite trace s, for which there is no finite trace r, such that $m \overset{sr}{\Longrightarrow} \mathsf{no}$, which is a contradiction. Observe that $\mathsf{f}(m) \in$ SHML. Then, from Lem. 5.10, the SHML formula $\mathsf{f}(m)$ is in EHML, and all of its subformulae can refute. Since m is sound for φ and sound and violation complete for $\mathsf{f}(m)$, it is the case that $\text{ACT}^\infty \setminus [\![\mathsf{f}(m)]\!] \subseteq \text{ACT}^\infty \setminus [\![\varphi]\!]$, and therefore $\mathsf{f}(m) \wedge \varphi \in$ SPHML and $[\![\mathsf{f}(m) \wedge \varphi]\!] = [\![\varphi]\!]$. □

Theorem 5.3 *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 $[\![\psi]\!] = [\![\varphi]\!]$.*

Proof A consequence of Lems. 5.8 and 5.11. □

5.4 Persistently informative monitorability, syntactically

We now give a syntactic characterisation of persistently informative monitorability. The reasoning is rather different from the one we employed for the previous fragments of RECHML, and relies on a *deterministic* form for RECHML.

We first introduce the *deterministic* fragment of RECHML and argue that all RECHML formulas can be determinised. This is a simple consequence of the expressive completeness of deterministic finite automata and deterministic parity automata in the domains of regular and ω-regular languages, respectively [39, 52].

We start by defining the deterministic fragment of RECHML (Def. 5.6). We continue by giving background on deterministic automata over finite, infinite, and finfinite traces (Def. 5.7). We show that every RECHML formula is equivalent to a deterministic automaton over finfinite traces (Lem. 5.12), and then we use this result to prove that every RECHML formula is equivalent to a deterministic one over finfinite traces (Lem. 5.13). This allows us to identify the persistently informatively monitorable formulas as certain deterministic formulas with special characteristics (Thm. 5.4).

Definition 5.6 The *deterministic* fragment DHML of RECHML is given by:

$$\varphi \in \text{DHML} ::= \mathsf{tt} \mid \mathsf{ff} \mid \bigwedge_{a \in \text{ACT}} [a]\varphi_a \mid \bigvee_{a \in \text{ACT}} \langle a \rangle \varphi_a \mid \mathsf{max}\, X.\varphi \mid \mathsf{min}\, X.\varphi \mid X.$$

In order to motivate the definition of this fragment, consider a formula $\varphi \in$ dHML and let ψ be one of its subformulae. Let s be the finite trace consisting of the modalities leading to an occurrence of ψ in φ. Then a finfinite trace sf satisfies φ if, and only if, f satisfies ψ. In contrast, for subformulae of a sHML formula, sf can only be made to violate the formula by a suffix f that falsifies the subfomula; the dual statement holds true for cHML. Since persistently informative monitorability depends on both violations and satisfactions, we turn to the deterministic fragment in Def. 5.6.

While the determinisation of both finite automata and ω-automata are standard, automata over the finfinite domain are not well-established. We define these automata and show that using determinisation procedures from the finite and the infinite domain, we can obtain, for any recHML formula φ, a deterministic automaton over finfinite words that recognises the traces satisfying φ. We then translate such automata into dHML.

The following definition recalls the definitions of deterministic automata over finite and infinite traces (words) and defines deterministic automata over finfinite traces.

Definition 5.7 A deterministic automaton is given by $\mathcal{D} = (Q, \Sigma, q_0, \delta, \Omega)$ where Q is a set of states, Σ is an alphabet, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to Q$ is a transition function and Ω is an acceptance condition, which depends on the type of the automaton.

For deterministic automata over *finite* traces (DFA), Ω is a subset $F \subseteq Q$; for deterministic automata over *infinite* traces (DPA), Ω is a priority assignment $\rho : Q \to I$ where I is a finite set of integer priorities; for deterministic automata over finfinite traces (DPFA), Ω is a pair of the form (F, ρ).

A run of an automaton $\mathcal{D}$ over a finite trace $s \in \Sigma^*$ is a sequence of states $\pi = \pi_0\pi_1 \cdots \pi_{|s|+1}$ of length $|s|+1$ such that $\pi_0 = q$ and $\pi_{i+1} = \delta(\pi_i, s[i])$. Similarly, a run of an automaton $\mathcal{D}$ over an infinite trace $t \in \Sigma^\omega$ is an infinite sequence of states $\pi = \pi_0\pi_1...$, such that $\pi_0 = q$ and $\pi_{i+1} = \delta(\pi_i, t[i])$. A run of a DFA over a finite word is accepting if the final state of the run is in F; a run of a DPA over an infinite word is accepting if the highest priority assigned by ρ to a state occuring infinitely often on the run is even; a run of a DPFA over a finfinite word is accepting if it is either finite and its final state is in F or it is infinite and the highest priority assigned by ρ to a state occuring infinitely often is even.

A deterministic automaton $\mathcal{D}$ accepts a word t if the (unique) run over t is accepting. The language recognised by the automaton, $L(\mathcal{D})$ is the set of traces that $\mathcal{D}$ accepts. ■

DFA are known to recognise all regular properties over finite traces while DPA recognise all ω-regular properties over infinite traces. We now argue that it follows that any recHML property φ is recognised by a DPFA.

Lemma 5.12 *For each* recHML *formula φ there is a DPFA that recognises the language of finfinite traces that satisfy φ.*

Proof The set of finite traces S_* that satisfy φ is a regular property of finite words, and therefore there is a DFA $\mathcal{D}_* = (Q, \textsc{Act}, q_0, \delta, F)$ that recognises S_*. Similarly, the set S_ω of infinite traces that satisfy φ is ω-regular, so there is a DPA $\mathcal{D}_\omega = (Q', \textsc{Act}, q_0', \delta', \rho)$ that recognises S_ω.

Let $\mathcal{D} = (Q \times Q', \textsc{Act}, (q_0, q_0'), \Delta, (F', \rho'))$ where $\Delta((q,q'),a) = (\delta(q,a), \delta(q',a))$ and $F' = F \times Q'$ and $\rho'(q,q') = \rho(q')$.

$\mathcal{D}$ recognises $[\![\varphi]\!]$. Indeed, $\mathcal{D}$ accepts a finite trace s if and only if the first component of its run is an accepting run over s in $\mathcal{D}_*$, and an infinite trace t if and only if the second component of its run is an accepting run over t in $\mathcal{D}_\omega$. □

Lemma 5.13 *For every* recHML *formula* φ*, there is an equivalent* dHML *formula* ψ*.*

Proof From Lem. 5.12, there is a DPFA $\mathcal{D} = (Q, \textsc{Act}, q_0, \delta, (F, \rho))$ that accepts exactly the traces that satisfy φ. We now show how to translate $\mathcal{D}$ into a dHML formula that is equivalent to φ.

We now consider all (finite) paths in $\mathcal{D}$ that start from q_0. For $k \geq 0$, states $q_1, q_2, \ldots, q_k \in Q$, and actions $a_1, a_2, \ldots, a_k \in \textsc{Act}$, $\varpi = q_0 a_1 q_1 a_2 q_2 \cdots a_k q_k$ is a path (for our purposes) of length k in $\mathcal{D}$, if

- for all states q_i, q_j, where $j > i$, if $q_i = q_j$, then there is some $i < l < j$, such that $\rho(q_l) > \rho(q_i)$; and
- for all $i < k$, $q_{i+1} = \delta(q_i, a_{i+1})$.

It is not hard to see, with a combinatorial argument, that $k \leq 2^{|Q|}$ (the highest priority can only occur once, the second highest twice, and the i^{th}-highest 2^{i-1} times). We use the notations $q_\varpi = q_k$ and $\varpi|_q = q_0 a_1 q_1 a_2 q_2 \cdots a_i q_i$, where $q_i = q$ is the last position where q appears in the path.

We then define a formula for each path $\varpi = q_0 a_1 q_1 a_2 q_2 \cdots a_k q_k$:

$$\varphi_\varpi = \begin{cases} \mathsf{max}\, X_\varpi.\ \bigwedge_{a \in \textsc{Act}} [a] g(\varpi, a), & \text{if } q_k \in F \text{ and } \rho(q_k) \text{ is even;} \\ \mathsf{min}\, X_\varpi.\ \bigwedge_{a \in \textsc{Act}} [a] g(\varpi, a), & \text{if } q_k \in F \text{ and } \rho(q_k) \text{ is odd;} \\ \mathsf{max}\, X_\varpi.\ \bigvee_{a \in \textsc{Act}} \langle a \rangle g(\varpi, a), & \text{if } q_k \notin F \text{ and } \rho(q_k) \text{ is even;} \\ \mathsf{min}\, X_\varpi.\ \bigvee_{a \in \textsc{Act}} \langle a \rangle g(\varpi, a), & \text{if } q_k \notin F \text{ and } \rho(q_k) \text{ is odd;} \end{cases}$$

where $g(\varpi, a) = \varphi_{\varpi a \delta(q,a)}$ if $\varpi a \delta(q,a)$ is a path, and $X_{\varpi|_{\delta(q,a)}}$ otherwise. Furthermore, we define ψ_ϖ to be such that $\varphi_\varpi = \mathsf{max}\, X_\varpi.\psi_\varpi$, or $\varphi_\varpi = \mathsf{min}\, X_\varpi.\psi_\varpi$.

Observe that the definition above is recursive, with maximal paths as base cases, and therefore for all ϖ, φ_ϖ is well defined. Furthermore, according to the above definition, a fixpoint variable appears only if it is marked by a subscript of a prefix of the corresponding path, and therefore it appears only in the scope of a (unique) formula that binds it.

We proceed to prove that $[\![\varphi_{q_0}]\!]$ is exactly the language of $\mathcal{D}$, *i.e.*, we show that for every finfinite trace f, $\mathcal{D}$ accepts f if and only if $f \in [\![\varphi_{q_0}]\!]$. We distinguish two cases.

Case 1: f is a finite trace. For this case, we consider an environment σ, such that for every path ϖ, $\sigma(X_\varpi) = [\![\varphi_\varpi, \sigma]\!]$, and we use induction on f to prove that for every path $\varpi = q_0 a_1 q_1 \cdots a_k q_k$, $f \in [\![\varphi_\varpi, \sigma]\!]$ if and only if the run of $\mathcal{D}$ from q_ϖ on f is an accepting run, and this suffices, because φ_{q_0} is a closed formula.

Case 2: f is an infinite trace. In this case, let $\pi = \pi_0 \pi_1 \cdots$ be the (infinite) run of $\mathcal{D}$ on f, where $q_0 = \pi_0$. Let $f = a_1 a_2 \cdots$, and for each $i \geq 0$, let $f_i = a_i a_{i+1} \cdots$. We can define the path-run $\pi' = \pi_0' \pi_1' \cdots$, where each π_i' is a path in $\mathcal{D}$, such that $\pi_0' = q_0$, and for all $i > 0$, if $\pi_{i-1}' \delta(q_{\pi_{i-1}'}, a_i)$ is a path, then

$\pi'_i = \pi'_{i-1}\delta(q_{\pi'_{i-1}}, a_i)$, and otherwise $\pi'_i = \pi'_{i-1}|_{\delta(q_{\pi'_{i-1}}, a_i)}$. Let q be such that $\rho(q)$ is the highest priority that appears infinitely often in the run.

We first assume that $\mathcal{D}$ accepts f (and therefore $\rho(q)$ is even), and we prove that $f \in [\![\varphi_{q_0}]\!]$. Let $I_0 \geq 0$ be such that $\pi_{I_0} = q$, and every priority that does not appear infinitely often in the path-run, only appears before position I_0. We proceed to prove the following claims:

Claim 1: $f_{I_0} \in [\![\varphi_{\pi'_{I_0}}, \sigma]\!]$. Since $\rho(q_{\pi'_{I_0}})$ is even, $\varphi_{\pi'_{I_0}}$ is a greatest fixpoint formula, and therefore, from its semantics, it suffices to find a set of traces S, such that $S \subseteq [\![\psi_{\pi'_{I_0}}, \sigma[X_{\pi'_{I_0}} \mapsto S]]\!]$. Let $S = \{f_i \mid i \geq I_0 \text{ and } \pi'_i = \pi'_{I_0}\}$. Let $I' > I \geq I_0$ be such that $\pi'_I = \pi'_{I'} = \pi'_{I_0}$. To prove the claim, it suffices to prove that $f_I \in [\![\varphi_{\pi'_I}, \sigma[X_{\pi'_{I_0}} \mapsto S]]\!]$. Note that $\rho(\pi'_I)$ is the greatest priority that appears from position I onward, and therefore all paths that appear in the path-run after position I are extensions of π'_I. Therefore, all $\varphi_{\pi'_i}$, where $i \geq I$ are subformulas of $\varphi_{\pi'_I}$. We show that for every $I \leq i < I'$, $f_i \in [\![\varphi_{\pi'_i}, \sigma[X_{\pi'_{I_0}} \mapsto S]]\!]$ and we use induction on $I' - i$. The base case is $i + 1 = I'$, and therefore $g(\varpi, a_i) = X_{\pi'_{I_0}}$, so $f_{i+1} \in S \subseteq [\![g(\varpi, a_i), \sigma[X_{\pi'_{I_0}} \mapsto S]]\!]$, yielding that $f_i \in [\![\varphi_{\pi'_i}, \sigma[X_{\pi'_{I_0}} \mapsto S]]\!]$. The inductive step is straightforward, after observing that $\varphi_{\pi'_i}$ is equivalent to $\psi_{\pi'_i}[\varphi_{\pi'_i}/X_{\pi'_i}]$, which, under $\sigma[X_{\pi'_{I_0}} \mapsto S]$ is equivalent to $\psi_{\pi'_i}$ (we have established that $X_{\pi'_{I_0}} \neq X_{\pi'_i}$).

Claim 2: for all $i \leq I_0$, $f_i \in [\![\varphi_{\pi'_i}, \sigma]\!]$. We can prove this by induction on $I_0 - i$. The base case is Claim 1 and the inductive steps are straightforward and similar to the above.

We now assume that $\mathcal{D}$ does not accept f (and therefore $\rho(q)$ is odd), and we prove that $f \notin [\![\varphi_{q_0}]\!]$. This case is similar to the above. □

We are ready to define the persistently informative fragment of recHML.

Definition 5.8 The persistently informatively monitorable fragment of recHML is pHML, which consists of all the formulas in dHML all of whose subformulas can refute or verify.

Theorem 5.4 *For $\varphi \in$ recHML, φ is persistently informatively monitorable if and only if there is some $\psi \in$ pHML such that $[\![\varphi]\!] = [\![\psi]\!]$.*

Proof Assume that $\varphi \in$ recHML is a persistently informatively monitorable. By Lem. 5.13, we can assume, without loss of generality, that $\varphi \in$ dHML. Furthermore, assume that unsatisfiable subformulas are replaced by ff and valid subformulas are replaced by tt. Towards a contradiction, assume that a subformula ψ of φ can neither refute or verify. Consider a sequence of modalities under the scope of which ψ is located and let s be the finite trace read off these modalities. Since φ is persistently informatively monitorable, it has a sound persistently informative monitor, and therefore, from Lem. 3.1, there is some r such that sr determines φ. Since ψ can neither refute nor verify, ψ is neither ff nor tt; since it is neither valid nor unsatisfiable, there are traces $srt \in [\![\varphi]\!]$ and $srt' \notin [\![\psi]\!]$, contradicting that sr determines φ.

For the other direction, consider $\varphi \in$ dHML all of whose subformulas can refute or verify. Then, for every trace s, we can find some r such that sr determines φ;

indeed r is the trace labelling the sequence of modalities leading to tt or ff. Hence φ is persistently monitorable, and we are done. □

This concludes our quest for syntactic characterisations of regular properties monitorable according to the different levels of our monitorability hierarchy. We now turn our attention to how existing notions of monitorability from the literature embed into this hierarchy, starting with *(co-)safety* properties.

6 Safety and Co-safety

The classic (and perhaps the most intuitive) definition of monitorability consists of (some variation of) *safety* properties [5, 9, 30, 37, 50, 53]. 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 [53] 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 Sec. 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[30, Def. 4] and, when restricted to infinite traces, to other work such as [9, 19, 37].

Definition 6.1 (Safety) A property $P \subseteq \textsc{Act}^{\infty}$ 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. 1.1. ■

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 [30], reachability [18]), is monitorable for satisfaction. That is, every trace that satisfies a co-safety property can be positively determined by a finite prefix.

Definition 6.2 (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. 1.1. ■

Example 6.1 "Eventually s *is reached", i.e.,* $\mathsf{F}\,\mathsf{s}$, 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.,* $\mathsf{G}\,\mathsf{F}\,\mathsf{s}$, 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 4.2 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. 1.1.

Theorem 6.1 *VCmp = Safe and SCmp = 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 (Lem. 3.2) and violation-complete (Def. 3.2) 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.* [5] already show the correspondence between violation (dually, satisfaction) monitorability over finfinite traces and properties expressible in sHML (dually, cHML). As a corollary of Thm. 6.1, we obtain a syntactic characterisation for the Safe and CoSafe monitorability classes; see Fig. 1.1.

6.1 Monitorability according to Falcone *et al.*

Falcone *et al.* [30] propose three definitions of monitorability (Definitions 16 and 17 in [30]) which they claim to coincide with safety, co-safety, and the union of safety and co-safety properties (Theorem 3 in [30]). We discuss this claim in more detail here, and argue that it does not hold. In brief, their definition deems all properties that are uniform over finite traces, such as "`success` infinitely often", or "the trace is finite" to be monitorable, not just safety and co-safety properties. In this appendix we recall Falcone *et al.*'s definitions and show that their definitions of monitorability include more than just safety and co-safety properties.

Remark 6.1 Falcone *et al.* present finfinite properties as a pair consisting of a set of finite traces and a set of infinite traces. Here we will speak of just one set, containing both finite and infinite traces.

The definition of monitorability proposed by Falcone *et al.* in [30] is parameterised by a truth domain, and a mapping of formulas into this domain. They then give a uniform condition that defines monitorability with respect to any truth-domain and its associated mapping. Here we focus on their monitorability with respect to the truth-domains $\{\mathsf{tt}, ?\}$, $\{\mathsf{ff}, ?\}$ and $\{\mathsf{tt}, \mathsf{ff}, ?\}$, which they claim correspond to co-safety, safety and their union, respectively.

Definition 6.3 (Property evaluation with respect to a truth-domain [30]) For each of three different verdict-domains and finfinite properties P ("r-properties" in their terminology), Falcone, Fernandez and Mournier define the following evaluation functions:

For $\mathbb{B} = \{\mathsf{ff}, ?\}$ and $s \in \textsc{Act}^*$:
$[\![P]\!]_{\mathbb{B}}(s) = \mathsf{ff}$ if $\forall f \in \textsc{Act}^\infty.\ sf \notin P$
$[\![P]\!]_{\mathbb{B}}(s) =?$ otherwise.
For $\mathbb{B} = \{\mathsf{tt}, ?\}$ and $s \in \textsc{Act}^*$:
$[\![P]\!]_{\mathbb{B}}(s) = \mathsf{tt}$ if $\forall f \in \textsc{Act}^\infty.\ sf \in P$
$[\![P]\!]_{\mathbb{B}}(s) =?$ otherwise.
For $\mathbb{B} = \{\mathsf{tt}, \mathsf{ff}, ?\}$ and $s \in \textsc{Act}^*$:
$[\![P]\!]_{\mathbb{B}}(s) = \mathsf{tt}$ if $s \in P$ and $\forall f \in \textsc{Act}^\infty.\ sf \in P$
$[\![P]\!]_{\mathbb{B}}(s) = \mathsf{ff}$ if $s \notin P$ and $\forall f \in \textsc{Act}^\infty.\ sf \notin P$
$[\![P]\!]_{\mathbb{B}}(s) =?$ otherwise. ■

Definition 6.4 (FFM-monitorability Definition 17, [30]) A property P is $\mathbb{B}$-monitorable over a truth domain $\mathbb{B}$ if for all $s, r \in \textsc{Act}^*$, if $s \in P$ and $r \notin P$, then $[\![P]\!]_{\mathbb{B}}(s) \neq [\![P]\!]_{\mathbb{B}}(r)$. ■

From this definition, it easily follows that any property P for which $P \cap \textsc{Act}^* = \emptyset$ or $\textsc{Act}^* \subseteq P$ is vacuously monitorable for any truth-domain, and evaluation function. However, not all such properties are safety or co-safety properties: "always eventually `success`" for instance is neither a safety nor a co-safety property.

We believe the critical points are Lemma 3 and Theorem 3 in [30], which do not hold. The proof of Lemma 3 in particular (Appendix 2.3) falsely claims that $P \cap \textsc{Act}^* = \emptyset$ or $\textsc{Act}^* \subseteq P$ implies that P is a safety or co-safety properties.

7 Pnueli and Zaks

The work on monitorability due to Pnueli and Zaks [47] is often cited by the RV community [15]. The often overlooked particularity of their definitions is that they only define monitorability of a property *with respect to a (finite) sequence.*

Definition 7.1 ([47]) 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 7.1 The property $(\mathsf{f} \wedge \mathsf{F\,r}) \vee (\mathsf{F\,G\,s})$ 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 $\forall$PZ- and $\exists$PZ-monitorability respectively. $\forall$PZ-monitorability is the more standard interpretation: it appears for example in [16, 30] where it is attributed to Pnueli and Zaks. However, the original intent seems to align more with $\exists$PZ-monitorability: in [47], 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 [21].

Definition 7.2 ($\forall$PZ-monitorability) A property P is (universally Pnueli–Zaks) $\forall$PZ-monitorable if it is s-monitorable for *all* finite traces s. The class of all $\forall$PZ-monitorable properties is denoted $\forall\mathsf{PZ}$. ■

Definition 7.3 ($\exists$PZ-monitorability) A property is (existentially Pnueli–Zaks) $\exists$PZ-monitorable if it is s-monitorable for *some* finite trace s, *i.e.,* if it is ε-monitorable. The class of $\exists$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 7.2 Consider the property *"Either* s *occurs before* f*, or* r *happens infinitely often"*, expressed in LTL fashion as $((\neg\mathsf{f})\,\mathsf{U}\,\mathsf{s}) \vee (\mathsf{G\,F\,r})$. This property is $\exists$PZ-monitorable because the trace s positively determines the property. However, it is *not* $\forall$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 6.1, there is no finite trace that determines the second conjunct positively or negatively. Property φ_{even} from Example 4.2 is $\forall$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 Defs. 7.2 and 7.3, it follows immediately that $\forall\mathsf{PZ} \subset \exists\mathsf{PZ}$.

Proposition 7.1 *All properties in $\mathsf{Safe} \cup \mathsf{CoSafe}$ are $\forall$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 Def. 6.1, 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 Def. 7.1. The case for $P \in \mathsf{CoSafe}$ is dual. □

Pnueli and Zaks, operationally. $\exists$PZ-monitorability coincides with informative monitorability: $\exists$PZ-monitorable properties are those for which some monitor can reach a verdict on some finite trace. For similar reasons, $\forall$PZ-monitorability coincides with persistently informative monitorability. See Fig. 1.1.

Theorem 7.1 $\exists\mathsf{PZ} = \mathsf{ICmp}$ *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 $s \in \textsc{Act}^*$. By Lem. 3.2, m_P is sound for P. By Def. 3.6 we need to show that there exists an f such that $\mathbf{acc}(m_P, sf)$ or $\mathbf{rej}(m_P, sf)$. From Defs. 7.1 and 7.2 we know that there is a finite r such that sr positively or negatively determines P. By Def. 3.2 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 $s \in \textsc{Act}^*$. By Defs. 7.1 and 7.2, we need to show that there is an extension of s that positively or negatively determines P. From Defs. 3.6 and 3.7, there exists some f such that $\mathbf{acc}(m_P, sf)$ or $\mathbf{rej}(m_P, sf)$. By Def. 3.1, 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 Def. 3.2, 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, focussing 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 $\textsc{Act}$. For instance, if $\textsc{Act}$ 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 $\text{ACT} = \{a\}$. 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 p but its extension a^ω does. On the other hand, when restricted to infinite traces, p 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 [16]. Falcone *et al.* [30] also have a definition of monitorability based on this idea (in addition to those discussed in Sec. 6.1). 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: $\mathsf{F\ G\ s}$ for the system from Example 1.1 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 [17] and attribute it to Pnueli and Zaks. However, unlike Falcone *et al.*, Pnueli and Zaks [47] 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 [38] (*a.k.a.* modal logic) is monitorable for both satisfaction and violation over infinite traces [5]. Furthermore, some properties, like $\{a^\omega\}$ over $\text{ACT} = \{a\}$, 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. 1.1 changes for the infinite domain is left for future work.

Havelund and Peled recently presented a related classification of infinitary properties [37]. 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.* [14] consider monitoring of properties over *finite* traces. In this domain, all properties are monitorable if, as is the case in [14], the end of a trace is *observable*; in this setting the question of monitorability is less relevant.

Monitorability parameterised by the domain Instead of considering finite, infinite of finfinite traces, we could equally consider monitorability with respect to any set of traces S. This could, for example, reflect some prior knowledge we have about the system. Then, the level of S-monitorability of a property will correspond to the guarantees that monitors can provide *assuming the execution is from* S. This approach is also called grey-box monitoring, as it no longer treats the system as a black box, and has been considered in [51] for hyperproperties.

Monitoring non-regular properties. Although we have focussed on the monitorability of regular properties, the monitorability hierarchy of Sec. 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 [14], in [31], 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 [24, 25] 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* [7, 30] is a criteria 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 1.1 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 Sec. 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: S. Lokam, R. Ramanujam (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: Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, *LNCS*, vol. 10803, pp. 203–220 (2018). URL `https://doi.org/10.1007/978-3-319-89366-2_11`
3. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Kjartansson, S.Ö.: Determinizing monitors for HML with recursion. CoRR **abs/1611.10212** (2016). URL `http://arxiv.org/abs/1611.10212`
4. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Kjartansson, S.Ö.: On the complexity of determinizing monitors. In: A. Carayol, C. Nicaud (eds.) Implementation and Application of Automata - 22nd International Conference, CIAA 2017, *LNCS*, vol. 10329, pp. 1–13. Springer (2017). URL `https://doi.org/10.1007/978-3-319-60134-2_1`
5. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Lehtinen, K.: Adventures in monitorability: From branching to linear time and back again. Proceedings of the ACM on Programming Languages **3**(POPL), 52:1–52:29 (2019). URL `https://dl.acm.org/citation.cfm?id=3290365`
6. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Lehtinen, K.: An operational guide to monitorability. In: P.C. Ölveczky, G. Salaün (eds.) Software Engineering and Formal Methods - 17th International Conference, SEFM 2019, Oslo, Norway, September 18-20, 2019, Proceedings, *Lecture Notes in Computer Science*, vol. 11724, pp. 433–453. Springer (2019). DOI 10.1007/978-3-030-30446-1_23. URL `https://doi.org/10.1007/978-3-030-30446-1_23`
7. 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). URL `https://doi.org/10.4230/LIPIcs.CONCUR.2018.34`
8. Aceto, L., Ingólfsdóttir, A., Larsen, K.G., Srba, J.: Reactive Systems: Modelling, Specification and Verification. Cambridge Univ. Press, New York, NY, USA (2007)
9. Alpern, B., Schneider, F.B.: Defining liveness. Information processing letters **21**(4), 181–185 (1985)
10. Arnold, A., Niwinski, D.: Rudiments of μ-calculus, *Studies in Logic and the Foundations of Mathematics*, vol. 146. North-Holland (2001)
11. Attard, D.P., Cassar, I., Francalanza, A., Aceto, L., Ingolfsdottir, A.: A runtime monitoring tool for actor-based systems. In: S. Gay, A. Ravara (eds.) Behavioural Types: From Theory to Tools, pp. 49–74. River Publishers (2017)
12. Attard, D.P., Francalanza, A.: A monitoring tool for a branching-time logic. In: Y. Falcone, C. Sánchez (eds.) Runtime Verification - 16th International Conference, RV 2016, *LNCS*, vol. 10012, pp. 473–481. Springer (2016). URL `https://doi.org/10.1007/978-3-319-46982-9_31`
13. Baier, C., Tinelli, C. (eds.): Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, *LNCS*, vol. 9035. Springer (2015)
14. Barringer, H., Rydeheard, D., Havelund, K.: Rule systems for run-time monitoring: from Eagle to RuleR. Journal of Logic and Computation **20**(3), 675–706 (2008)
15. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to runtime verification. In: E. Bartocci, Y. Falcone (eds.) Lectures on Runtime Verification - Introductory and Advanced Topics, *LNCS*, vol. 10457, pp. 1–33. Springer (2018). URL `https://doi.org/10.1007/978-3-319-75632-5_1`
16. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. Journal of Logic and Computation **20**(3), 651–674 (2010)
17. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology **20**(4), 14:1–14:64 (2011). URL `http://doi.acm.org/10.1145/2000799.2000800`
18. Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P.: Systems and Software Verification: Model-checking Techniques and Tools. Springer Science & Business Media (2013)
19. Chang, E.Y., Manna, Z., Pnueli, A.: Characterization of temporal property classes. In: W. Kuich (ed.) Automata, Languages and Programming, 19th International Colloquium, ICALP 1992, *LNCS*, vol. 623, pp. 474–486. Springer (1992). URL `https://doi.org/10.1007/3-540-55719-9_97`
20. Chen, F., Rosu, G.: Mop: an efficient and generic runtime verification framework. In: R.P. Gabriel, D.F. Bacon, C.V. Lopes, G.L.S. Jr. (eds.) Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and

Applications, OOPSLA 2007, pp. 569–588. ACM (2007). URL `https://doi.org/10.1145/1297027.1297069`

21. 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 (2018)
22. Cini, C., Francalanza, A.: An LTL proof system for runtime verification. In: Baier and Tinelli [13], pp. 581–595. URL `https://doi.org/10.1007/978-3-662-46681-0_54`
23. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT press (1999)
24. Convent, L., Hungerecker, S., Leucker, M., Scheffel, T., Schmitz, M., Thoma, D.: TeSSLa: Temporal Stream-Based Specification Language. In: Formal Methods: Foundations and Applications - 21st Brazilian Symposium, SBMF 2018, *LNCS*, vol. 11254, pp. 144–162 (2018). DOI 10.1007/978-3-030-03044-5_10
25. D'Angelo, B., Sankaranarayanan, S., Sánchez, C., Robinson, W., Finkbeiner, B., Sipma, H.B., Mehrotra, S., Manna, Z.: Lola: Runtime monitoring of synchronous systems. In: 12^{th} International Symposium on Temporal Representation and Reasoning (TIME'05), pp. 166–174. IEEE Computer Society Press (2005)
26. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: F. Rossi (ed.) IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, pp. 854–860. IJCAI/AAAI (2013). URL `http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997`
27. Decker, N., Leucker, M., Thoma, D.: jUnit^{rv}-adding runtime verification to jUnit. In: NASA Formal Methods, 5th International Symposium, NFM, *LNCS*, vol. 7871, pp. 459–464 (2013). URL `https://doi.org/10.1007/978-3-642-38088-4_34`
28. Diekert, V., Gastin, P.: First-order definable languages. In: Logic and Automata: History and Perspectives, Texts in Logic and Games, pp. 261–306. Amsterdam University Press (2008)
29. Diekert, V., Leucker, M.: Topology, monitorable properties and runtime verification. Theoretical Computer Science **537**, 29–41 (2014). DOI 10.1016/j.tcs.2014.02.052
30. Falcone, Y., Fernandez, J.C., Mounier, L.: What can you verify and enforce at runtime? International Journal on Software Tools for Technology Transfer **14**(3), 349–382 (2012)
31. Ferrère, T., Henzinger, T.A., Saraç, N.E.: A theory of register monitors. In: A. Dawar, E. Grädel (eds.) Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, pp. 394–403. ACM (2018). URL `https://doi.org/10.1145/3209108.3209194`
32. Francalanza, A.: A Theory of Monitors (Extended Abstract). In: Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS, Eindhoven, The Netherlands, *LNCS*, vol. 9634, pp. 145–161 (2016)
33. Francalanza, A.: Consistently-detecting monitors. In: 28^{th} International Conference on Concurrency Theory (CONCUR), *LIPIcs*, vol. 85, pp. 8:1–8:19. Schloss Dagstuhl (2017). DOI 10.4230/LIPIcs.CONCUR.2017.8
34. Francalanza, A., Aceto, L., Achilleos, A., Attard, D.P., Cassar, I., Monica, D.D., Ingólfsdóttir, A.: A Foundation for Runtime Monitoring. In: Runtime Verification - 17th International Conference, RV 2017, *LNCS*, vol. 10548, pp. 8–29. Springer (2017). URL `https://doi.org/10.1007/978-3-319-67531-2_2`
35. Francalanza, A., Aceto, L., Ingólfsdóttir, A.: Monitorability for the Hennessy-Milner logic with recursion. Formal Methods in System Design **51**(1), 87–116 (2017). URL `https://doi.org/10.1007/s10703-017-0273-z`
36. Francalanza, A., Seychell, A.: Synthesising Correct concurrent Runtime Monitors. Formal Methods in System Design (FMSD) **46**(3), 226–261 (2015). URL `http://dx.doi.org/10.1007/s10703-014-0217-9`
37. Havelund, K., Peled, D.: Runtime Verification: From Propositional to First-Order Temporal Logic. In: Runtime Verification - 18th International Conference, RV 2018, Limassol, Cyprus, November 10-13, 2018, Proceedings, *LNCS*, vol. 11237, pp. 90–112. Springer (2018). URL `https://doi.org/10.1007/978-3-030-03769-7_7`
38. Hennessy, M., Milner, R.: Algebraic Laws for Nondeterminism and Concurrency. Journal of the ACM **32**(1), 137–161 (1985). DOI 10.1145/2455.2460
39. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation. Acm Sigact News **32**(1), 60–65 (2001)
40. Kozen, D.C.: Results on the propositional μ-calculus. Theoretical Computer Science **27**, 333–354 (1983)

41. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Formal Methods in System Design **19**(3), 291–314 (2001)
42. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. Journal of the ACM **47**(2), 312–360 (2000)
43. Larsen, K.G.: Proof systems for satisfiability in Hennessy-Milner logic with recursion. Theoretical Computer Science **72**(2), 265 – 288 (1990). DOI http://dx.doi.org/10.1016/0304-3975(90)90038-J
44. Laurent, J., Goodloe, A., Pike, L.: Assuring the Guardians. In: Runtime Verification (RV), *LNCS*, vol. 9333, pp. 87–101 (2015)
45. Manna, Z., Pnueli, A.: Completing the temporal picture. Theoretical Computer Science **83**(1), 97–130 (1991). DOI 10.1016/0304-3975(91)90041-Y
46. Neykova, R., Bocchi, L., Yoshida, N.: Timed runtime monitoring for multiparty conversations. Formal Aspects of Computing **29**(5), 877–910 (2017). URL `https://doi.org/10.1007/s00165-017-0420-8`
47. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: J. Misra, T. Nipkow, E. Sekerinski (eds.) FM 2006: Formal Methods, 14th International Symposium on Formal Methods, *LNCS*, vol. 4085, pp. 573–586. Springer (2006). URL `https://doi.org/10.1007/11813040_38`
48. Reger, G., Cruz, H.C., Rydeheard, D.E.: MarQ: Monitoring at runtime with QEA. In: Baier and Tinelli [13], pp. 596–610. URL `https://doi.org/10.1007/978-3-662-46681-0_55`
49. Rosu, G.: On safety properties and their monitoring. Scientific Annals of Computer Science **22**(2), 327–365 (2012)
50. Schneider, F.B.: Enforceable security policies. ACM Transactions on Information and System Security **3**(1), 30–50 (2000)
51. Stucki, S., Sánchez, C., Schneider, G., Bonakdarpour, B.: Gray-box monitoring of hyperproperties. In: M.H. ter Beek, A. McIver, J.N. Oliveira (eds.) Formal Methods – The Next 30 Years, pp. 406–424. Springer International Publishing, Cham (2019)
52. THOMAS, W.: Chapter 4 - automata on infinite objects. In: J.V. LEEUWEN (ed.) Formal Models and Semantics, Handbook of Theoretical Computer Science, pp. 133 – 191. Elsevier, Amsterdam (1990). DOI https://doi.org/10.1016/B978-0-444-88074-1.50009-3. URL `http://www.sciencedirect.com/science/article/pii/B9780444880741500093`
53. Viswanathan, M., Kim, M.: Foundations for the run-time monitoring of reactive systems - fundamentals of the MaC language. In: Z. Liu, K. Araki (eds.) Theoretical Aspects of Computing - ICTAC 2004, First International Colloquium, *LNCS*, vol. 3407, pp. 543–556. Springer (2004). URL `https://doi.org/10.1007/978-3-540-31862-0_38`
54. Wolper, P.: Temporal logic can be more expressive. Information and Control **56**(1/2), 72–99 (1983). URL `https://doi.org/10.1016/S0019-9958(83)80051-5`