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, project FouCo, funded by the EU H020 research and innovation programme under the Marie Skłodowska-Curie grant agreement №:892704, 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, 24, 32, 38, 50, 53]. 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, 53] 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, 21, 30, 51, 54], 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* [35, 36, 39, 48], 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, 38], 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 [38].

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, 33, 38, 40, 53]. 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 work on defining monitorability, by Falcone *et al.* [33], is inconsistent.

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 [53] (discussed in Section 8), this property is monitorable. However, it is not monitorable according to others, including Schneider [57], Viswanathan and Kim [61], 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 in 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 Figure 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 [16,17,40] (*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.

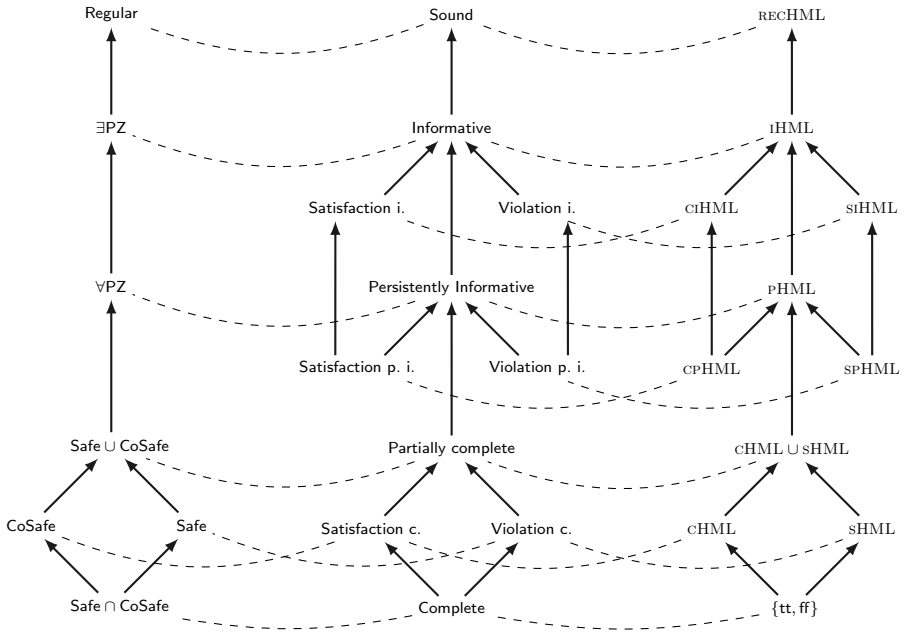

Fig. 1.1 The Monitorability Hierarchy of Regular Properties over Finfinite Traces

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 Figure 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 (Theorem 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

Hennessy–Milner logic with recursion, recHML [8, 46] (a variant of the modal μ-calculus [43]) interpreted over finfinite traces—see Figure 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 [37, 38] 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 [33]. 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.

However, since this is not the only interesting domain, we also consider how the set of traces considered affects the syntactic characterisations of different monitorability classes. We show that the syntactic characterisations for violation and satisfaction completeness are robust accross *all* linear domains. We also show that while this is not the case for informative and persistently informative monitorability, their syntactic characterisations are still valid for the important domain of infinite traces.

Finally, we show that the proposed hierarchy accounts for existing notions of monitorability. See Figure 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), also identified by Bauer, Leucker and Schallhart [18], to persistently informative monitorability. We also show that the definitions of monitorability proposed by Falcone *et al.* [33], contrary to their claim, do *not* coincide with safety and co-safety properties. To summarise, our principal contributions are:

1. A unified operational perspective on existing notions of monitorability, clarifying what operational guarantees each provides, see Theorems 3.1, 7.1 and 8.1;
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 Theorems 5.2 and 5.3, as well as proofs of robustness and non-robustness accross domains for each of these characterisations.

This article extends the conference version [6]. The main technical novelty here is the logical characterisation of persistently informative monitorability and the proofs and counter-examples of robustness accross domains for all the syntactic characterisations. 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

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

logical characterisations). We have also added detailed proofs, extended examples, added more detailed discussions of related work and improved explanations.

Roadmap. We start with defining notation for traces and properties in the finfinite domain in Section 2. We then define the monitorability hierarchy for properties over finfinite traces in Section 3, and instantiate it with concrete operational semantics in Section 4 for regular properties. In Section 5 we give syntactic characterisations of each level of our hierarchy. In Section 6 we study how robust each of these syntactic characterisations is if the domain changes, for example to the infinite, rather than finfinite, domain. In Sections 7 and 8 we show how existing notions of monitorability embed into our hierarchy and discuss an error in Falcone *et al.*'s notion of monitorability. Finally, before concluding, in Section 9 we discuss other notions of monitorability and how changing various aspects of the framework, such as the alphabet, 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 $P_{\text{inf}} \subseteq \textsc{Act}^\omega$ [60].

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 Section 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 Figure 1.1. Section 4 will then provide operational semantics to this hierarchy, in the setting of regular properties.

Note that, as we show in the second half of this manuscripts, the definitions that arise naturally in this general framework cover various definitions that have been presented in the literature, depicted in the left part of Figure 1.1.

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 a computational entity, m, 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 assume two postulates. Firstly, an acceptance or rejection verdict has to be based on a finite prefix of a trace because we target online monitors that observe an (ever increasing) trace of events generated by the running system under scrutiny. In Definition 3.1.1 verdicts are thus given for *incomplete* traces. Secondly, verdicts must be *irrevocable*, Definition 3.1.2. Without this second requirement, verdicts would become ephemeral (and not dependable, since they could change when more events from the running system are observed). These postulates make explicit two features shared by most monitorability definitions in the literature. On a technical level, verdict irrevocability allows us to extend acceptances and rejections to the continuation of the trace prefix (which can be infinite). Put differently, a monitor m can be abstractly conceived as an entity consisting of two predicates, **acc** and **rej**, defined over the *finfinite* trace domain as follows.

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 $\exists s \in \textsc{Act}^* \cdot \big(s \preceq f \text{ and } \mathbf{acc}(m, s)\big)$ and
   - $\mathbf{rej}(m, f)$ implies $\exists s \in \textsc{Act}^* \cdot \big(s \preceq f \text{ and } \mathbf{rej}(m, s)\big)$;
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 The set of finite automata does not satisfy the requirements of a monitoring system as defined in Definition 3.1 because (i) they do not process infinite traces and, more importantly, (ii) their verdicts can be revoked since they allow transitions from final to non-final states.[3] 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 $C \subseteq 2^{\text{Act}^\infty}$ if for every $P \in C$ there is a monitor $m_P \in M$ such that

(i) $\mathbf{acc}(m_P, f)$ iff trace $f \in \text{Act}^\infty$ has a prefix that positively determines P;
(ii) $\mathbf{rej}(m_P, f)$ iff trace $f \in \text{Act}^\infty$ has a prefix that negatively determines P. ■

In Section 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.

Remark 3.2 Working with an abstract maximal monitoring system allows us to build a hierarchy over all properties of finfinite traces, regardless of their computational complexity. However, considering the monitoring system as a parameter of monitorability enables this hierarchy to also account for questions of computability and resource usage, which become particularly relevant for non-regular properties. For instance, when monitoring context-free languages, it might be practical, from an implementation perspective, to restrict the monitors M to *deterministic* automata with a stack and irrevocable verdicts, rather than nondeterministic ones, to avoid excessive memory overheads associated with keeping track of several stacks. However, deterministic pushdown monitors are not powerful enough to make a *maximal* monitoring system for context-free languages, because pushdown automata are not in general determinisable. Then, the monitorability classes will differ, according to the computational power of the monitors; parameterising the hierarchy with the monitoring system allows us to account for these variations.

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 $f \notin P$. ■

Soundness is important for a number of reasons. For instance, it prevents monitor inconsistency which is arguably an intrinsic quality that is expected of any sensible monitor (Definition 3.2 does not preclude inconsistent monitors). We here adapt the definition of monitor consistency [5, Def. 3.2] to the present setting.

[3] One could restrict the class of automata used, but then would also need to show that closure properties are preserved, the proof of which does not seem immediate.

Definition 3.4 (Consistency) A monitor m is consistent if for all $t \in \text{Act}^\infty$:

- $\mathbf{acc}(m,t)$ implies that $\mathbf{rej}(m,t)$ does not hold.[4]
- $\mathbf{acc}(m,t)$ implies that $\mathbf{rej}(m,t)$ does not hold. ■

Lemma 3.1 *If a monitor m is sound for some property P then it is consistent.*

Proof Pick a monitor m that is sound for some property P and assume, towards a contradiction, that for some $t \in \text{Act}^\infty$, we have both $\mathbf{acc}(m,t)$ and $\mathbf{rej}(m,t)$. By soundness, this implies both $f \in P$ and $f \notin P$, which is clearly a contradiction. □

Lemma 3.2 *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 some $s \in \text{Act}^*$ where $\mathbf{acc}(m,s)$ and pick some $f \in \text{Act}^\infty$. By Definition 3.1.2 and $\mathbf{acc}(m,s)$ we know that $\mathbf{acc}(m,sf)$ and by soudness we obtain $sf \in P$. □

The following lemma explains why a maximal monitoring system subsumes other systems *wrt.* soundness.

Lemma 3.3 *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 For the first clause, pick a $f \in \text{Act}^\infty$ such that $\mathbf{acc}(m_P,f)$. By Definition 3.2, there exists some prefix $s \preceq f$ that positively determines P. Since f is an extension of s, to follows that $f \in P$. The case for $\mathbf{rej}(m_P,f)$ is analogous.

For the second clause, pick a finfinite trace f such that $\mathbf{acc}(m,f)$. By Definition 3.1 we know there exists some finite prefix, s where $s \preceq f$, such that $\mathbf{acc}(m,s)$. By Lemma 3.2 we know that s positively determines P. By Definition 3.2 we deduce that $\mathbf{acc}(m_P,s)$ and by $s \preceq f$ and Definition 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. Our tenet is that soundness is not negotiable. The dual requirement to soundness, *i.e., completeness*, entails that the monitor detects *all* violating and satisfying traces.

Definition 3.5 (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. ■

[4] When **acc** and **rej** are viewed as predicated, "$\mathbf{rej}(m,t)$ does not hold" means that either $\neg\mathbf{rej}(m,t)$ or that it is undefined.

Unfortunately, as shown now in Proposition 3.1, completeness is only possible for trivial properties in the finfinite domain; in the infinite domain more properties *are* completely monitorable—see Section 9.

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

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

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

Definition 3.6 (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 makes explicit the relation between the monitorability classes of Definition 3.6 and finite prefixes that determine a property, which are also called good and bad prefixes [44].

Lemma 3.4 *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 Definition 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 Definition 3.6, 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 3.7 (Informative Monitors[5]) A monitor m is:

[5] These are *not* related to the *informative prefixes* from [44] or to *persistence* from [55].

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

Definition 3.8 (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.
- A property P is informatively monitorable if there is an 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 Definition 3.1, it must accept some prefix $s \preceq f$. Again, by Definition 3.1, all continuations, including $s\mathsf{r}^\omega$, 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 Definitions 3.6 and 3.8 form the inclusion hierarchy depicted in Figure 1.1(middle).*

Proof The only non-trivial inclusion to show from Figure 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 Definition 3.5 we have $\mathbf{rej}(m_P, sf)$. Otherwise, $sf \in P$ for each f, meaning that s positively determines P, and by Definition 3.2 we have $\mathbf{acc}(m_P, sf)$. By Definition 3.7, we deduce that m_P is persistently informative since $\forall s \exists f \cdot \mathbf{acc}(m_P, sf)$ or $\mathbf{rej}(m_P, sf)$. Thus, by Definition 3.8, 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, one can simply observe that tt is not informatively monitorable for violation whereas ff is not informatively monitorable for satisfactions. ■

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 Hennessy–Milner logic with recursion [46], recHML, to represent regular properties. This is a reformulation of the modal μ-calculus [43], 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. This logic has deep connections to program equivalence theories [8, 41, 58] and has also been used to model-check systems using industry-strength verifiers [28, 47].

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 Definition 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 Section 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 following grammar, which assumes a countable set of logical variables $X, Y \in \text{LVar}$.

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

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 order of precedence of operators is, as usual: the existential and universal modal operators, conjunctions, disjunctions, and fixpoint operators. The semantics is given by a function $[\![-]\!]$, which 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. The function $[\![-]\!]$ is defined in the following:

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

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 [45]. In the discussions we occasionally treat formulae, φ, as the properties they denote, $[\![\varphi]\!]$.

LTL [25] 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, 40]). Our choice of logic, recHML, is not limiting in this regard because it is well known [43, 62] that LTL can be translated into recHML. There are other reasonable and equally expressive choices for the logic (see, for instance, [56]), but we chose recHML as it is convenient for synthesizing monitors as we see later in this section, and for identifying monitorable syntactic fragments as we do in Section 5.

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

$$
\begin{array}{lll}
\mathsf{X}\,\varphi \stackrel{\text{def}}{=} \bigvee_{a \in \textsc{Act}} \langle a\rangle\varphi & \varphi\,\mathsf{U}\,\psi \stackrel{\text{def}}{=} \mathsf{min}\, Y.\big(\psi \vee (\,\varphi \wedge \mathsf{X}\; Y)\big) & \mathsf{F}\,\varphi \stackrel{\text{def}}{=} \mathsf{tt}\,\mathsf{U}\,\varphi \\
\varphi\,\mathsf{R}\,\psi \stackrel{\text{def}}{=} \mathsf{max}\, Y.\big((\,\psi \wedge \varphi\,) \vee (\,\psi \wedge \mathsf{X}\; Y)\big) & & \mathsf{G}\,\varphi \stackrel{\text{def}}{=} \mathsf{ff}\,\mathsf{R}\,\varphi
\end{array}
$$

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 [62].

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 [62], 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 $\neg\mathsf{s} \wedge \mathsf{G}(\mathsf{s} \Leftrightarrow \mathsf{X}\neg\mathsf{s})$ 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 s^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:

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

Remark 4.1

More broadly, recHML captures all (ω-)regular properties, while LTL can only express properties recognised by counter-free Büchi automata [31]. 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, [29] 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.

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, translate it into 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, (the trace should not terminate there) in line with Example 4.1.

In the sequel, we use the following classical result for recHML-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 (Lemma 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 $[\![\varphi]\!] \cap \textsc{Act}^\omega$.

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

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

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

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

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

Fig. 4.1 Monitor Syntax and Labelled-Transition Semantics

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 [5,38], summarised in Figure 4.1 (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 $\odot \in \{\oplus, \otimes\}$. Composition with $\oplus$ corresponds to disjunctive parallelism, which reaches a positive verdict whenever any of its components reaches a positive verdict, while $\otimes$ is conjunctive in that is favours the negative verdict. There are three distinct *verdict* states, yes, no, and end, although only the first two are relevant to monitorability. The syntax in Figure 4.1 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 Act, which is comprised of the monitor states, or monitors, and a transition relation. The set of monitor states, Mon, and the monitor transition relation, $\longrightarrow \subseteq (\textsc{Mon} \times (\textsc{Act} \cup \{\tau\}) \times \textsc{Mon})$, are defined in Figure 4.1. 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 $m \overset{\mu}{\Longrightarrow} n$ for $m \Longrightarrow \cdot \xrightarrow{\mu} \cdot \Longrightarrow n$, where $\mu \in (\textsc{Act} \cup \{\tau\})$. We write sequences of transitions $m \overset{a_1}{\Longrightarrow} \cdots \overset{a_k}{\Longrightarrow} n$ as $m \overset{s}{\Longrightarrow} n$, 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, 35, 36, 38]. 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 $s = a_1 \dots a_k$ 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:

- $\mathbf{rej}(m, s)$ and say that m *rejects* when $m \stackrel{s}{\Longrightarrow} \mathsf{no}$
- $\mathbf{acc}(m, s)$ and say that m *accepts* when $m \stackrel{s}{\Longrightarrow} \mathsf{yes}$.

Similarly, for $t \in \textsc{Act}^\omega$, we write

- $\mathbf{rej}(m, t)$ if there exist $s \in \textsc{Act}^*$ and $u \in \textsc{Act}^\omega$ where $t{=}su$ and m rejects s.
- $\mathbf{acc}(m, t)$ if there exist $s \in \textsc{Act}^*$ and $u \in \textsc{Act}^\omega$ where $t{=}su$ and m 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 Lemma 4.3 assures us that regular monitors satisfy the conditions to be a monitoring system, given in Definition 3.1.

Lemma 4.3 (Verdict Persistence, [5, 38]) *For all verdicts v, it is the case that $v \stackrel{s}{\Longrightarrow} m$ 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 *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* Mon *is verdict equivalent to a deterministic regular monitor.*

Due to Theorem 4.1, we can assume that every monitor in 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 $\mathbf{acc}(m, s)$ iff $s \in L$ and $\mathbf{rej}(m, s)$ iff $s \in L'$.*

We use the formula synthesis function from regular monitors to formulae defined in [5, 38] (we assume a bijection between logical variables, X, and monitor variables, x, that we leave implicit):

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

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

From properties such as Lemma 4.3 is not hard to see that this operational framework satisfies the conditions for a monitoring system of Definition 3.1. The monitoring system of Figure 4.1 is also maximal for regular properties, according to Definition 3.2. This concrete instance thus demonstrates the realisability of the definitions in Section 3.

Theorem 4.4 *For all $\varphi \in$ recHML, 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 Lemma 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 Theorem 4.2. □

As a corollary of Theorem 4.4, from Lemma 3.2 we deduce that for any arbitrary monitoring system $(M, \mathbf{acc}, \mathbf{rej})$, if $m \in M$ is sound for some $\varphi \in$ recHML, then there is a monitor $n \in$ Mon from Figure 4.1 that accepts (*resp.*, rejects) all traces f that m accepts (*resp.*, rejects).

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

Proof By Theorem 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 $\mathbf{acc}(m, s)$ (*resp.*, $\mathbf{rej}(m, s)$) for some finite trace s, then, due to the soundness of m, $s \in [\![\varphi]\!]$ (*resp.*, $s \notin [\![\varphi]\!]$), and therefore, from Lemma 3.2, s positively (*resp.*, negatively) determines φ. By the properties of n, we have that $\mathbf{acc}(n, s)$ (*resp.*, $\mathbf{rej}(n, s)$). □

In the sequel, we thus assume (Mon, $\mathbf{acc}, \mathbf{rej}$) from Figure 4.1 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 recHML. 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:*

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

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 Theorem 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].

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

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

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

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, then there is a regular monitor that is sound and informatively rejecting for φ.*
– *If $\varphi \in$ ciHML, then there is a regular monitor that is sound and 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 Lemma 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 Theorem 4.4. □

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

Proof If m is sound and informatively accepting for φ, then by Lemma 3.2, 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 for satisfaction if and only if there is some $\psi \in$ 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 Lemmata 5.2 and 5.3. □

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 [44] for how to construct these formulas; this method has to be adapted a little in the finfinite domain, but this is outside the scope of the present work. ■

The maximality results of Theorems 5.1 and 5.2 permit tool constructors 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. Theorems 5.1 and 5.2 also serve as a syntactic check to determine when a property is monitorable (according to the monitorability classes in Figure 1.1).

Example 5.2 The formula $\varphi = \mathsf{min}\ X.\mathsf{max}\ Y.(\langle a\rangle\mathsf{tt} \vee \langle b\rangle\mathsf{tt} \vee X) \notin$ ciHML, but it is equivalent to $\varphi_1 = \varphi \vee \langle a\rangle\mathsf{tt} \in$ ciHML, to $\varphi_2 = \varphi \vee (\langle a\rangle\mathsf{tt} \vee \langle b\rangle\mathsf{tt}) \in$ ciHML, and to $\varphi_3 = \mathsf{min}\ X.(\langle a\rangle\mathsf{tt} \vee \langle b\rangle\mathsf{tt} \vee X) \in$ cHML. The formulas φ_1 and φ_2 are evidence that φ is informatively monitorable, while φ_3 is evidence that φ is monitorable for satisfactions.

In Example 5.2, if one uses φ_1 to synthesize a monitor for φ, the resulting monitor would detect the satisfying traces that start from a; starting from φ_2, the synthesized monitor would also detect the satisfying traces that start with b; finally, the monitor that is synthesized from φ_3 is satisfaction complete for φ. Thus, we observe that, depending on the form of the given formula, the monitor synthesis that results from these syntactic characterisations may not always yield monitors that detect all possible satisfactions or violations. However, Theorem 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.

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 formulas forces modal subformulas 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:

$$\varphi, \psi, \varphi_a \in \text{EHML} ::= \mathsf{tt} \mid \mathsf{ff} \mid \mathsf{min}\, X.\varphi \mid \mathsf{max}\, X.\varphi \mid X \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \bigvee_{a \in \text{ACT}} \langle a \rangle \varphi_a \mid \bigwedge_{a \in \text{ACT}} [a]\varphi_a . \quad \blacksquare$$

Example 5.3 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 subformulas (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
  - ψ 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 5.4 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 subformulas of $\mathsf{max}\, X.([\mathsf{s}]X \wedge [\mathsf{f}]\mathsf{ff} \wedge [\mathsf{r}]\mathsf{ff})$ can refute. ■

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

Definition 5.4 We define the fragments SPHML and CPHML as:

$$\text{SPHML} = \left\{ \varphi_1 \wedge \varphi_2 \in \text{RECHML} \;\middle|\; \begin{array}{l} \varphi_1 \in \text{SHML} \cap \text{EHML} \text{ and every} \\ \text{subformula of } \varphi_1 \text{ can refute} \end{array} \right\}$$

$$\text{CPHML} = \left\{ \varphi_1 \vee \varphi_2 \in \text{RECHML} \;\middle|\; \begin{array}{l} \varphi_1 \in \text{CHML} \cap \text{EHML} \text{ and every} \\ \text{subformula of } \varphi_1 \text{ can verify} \end{array} \right\} \quad \blacksquare$$

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

The proof of Theorem 5.3 can be found in Appendix A. The intuition is that every finite trace s can be extended to one that violates or satisfies, respectively, the φ_2 part in Definition 5.4. On the other hand, a formula that is persistently informatively monitorable for satisfaction or violation must have a sound persistently informative for satisfaction or violation, respectively, monitor, from which we can synthesize φ_2.

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 [42, 60].

We start by defining the deterministic fragment of recHML (Definition 5.5). We continue by giving background on deterministic automata over finite, infinite, and finfinite traces (Definition 5.6). We show that every recHML formula is equivalent to a deterministic automaton over finfinite traces (Lemma 5.4), and then we use this result to prove that every recHML formula is equivalent to a deterministic one over finfinite traces (Lemma 5.5). This allows us to identify the persistently informatively monitorable formulas as certain deterministic formulas with special characteristics (Theorem 5.4).

Definition 5.5 The *deterministic* fragment dHML of recHML is given by:

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

In order to motivate the definition of this fragment, consider a formula $\varphi \in$ dHML. By induction on the length of a finite trace s, we can see that for every s, there is a unique, up to unfolding, subformula ψ of φ, such that for every finfinite trace f, sf satisfies φ if, and only if, f satisfies ψ. Conversely, by structural induction on ψ, for every subformula ψ (after unfolding) of φ, there is a finite trace s, which intuitively is formed by the sequence of actions in the modalities of φ that leads to an occurrence of ψ in φ such that every finfinite trace sf satisfies φ if, and only if, f satisfies ψ. In contrast, for a sHML formula φ the violation of a subformula can only yield the violation of φ, while the satisfaction of a subformula often does not yield the satisfaction of φ, and dually for cHML. Since persistently informative monitorability depends on both violations and satisfactions, we turn to the deterministic fragment of Definition 5.5.

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 5.6 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 occurring 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 occurring 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 5.4 *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 $\mathcal{D}_* = (Q, \text{ACT}, q_0, \delta, F)$ that recognises S_*. Similarly, the set S_ω of infinite traces that satisfy φ is ω-regular, so there is a DPA $\mathcal{D}_\omega = (Q', \text{ACT}, q_0', \delta', \rho)$ that recognises S_ω.

Let $\mathcal{D} = (Q \times Q', \text{ACT}, (q_0, q_0'), \Delta, (F', \rho'))$ 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 5.5 *For every* RECHML *formula φ, there is an equivalent* DHML *formula ψ.*

Proof From Lemma 5.4, there is a DPFA $\mathcal{D} = (Q, \text{ACT}, q_0, \delta, (F, \rho))$ 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 $a_1, a_2, \ldots, a_k \in \text{Act}$, $\varpi = q_0 a_1 q_1 a_2 q_2 \cdots a_k q_k$ 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 $\varpi = q_0 a_1 q_1 a_2 q_2 \cdots a_k q_k$:

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

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 5.7 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 Lemma 5.5, 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 Lemma 3.2, 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. □

6 The robustness of syntactic fragments

So far, we have assumed that executions can be finite, as well as infinite. However, in some settings one may expect all executions to be, in fact, infinite, or, on the contrary, that all executions eventually terminate. Furthermore, we may also have access to prior knowledge about the system that affects the domain of executions to consider. In such cases, guarantees on monitor behaviour need only be restricted

to the domain of interest and it is natural to wonder how our syntactic hierarchy of monitorability adapts to different domains.

First, observe that a smaller domain means less stringnent requirements on monitors: in particular, monitors don't have to be sound with respect to executions excluded by the domain of study. In the infinite domain, for example, this enables monitors to reach verdicts that they could not reach in the finfinite domain. Indeed, modal properties, such as "`Initialise` occurs within the first 10 events", which are determined by a prefix of bounded length, are all monitorable both for violations and satisfactions in the infinite domain, but only persistently informatively monitorable in the finfinite domain.

Second, note that formulas that are not equivalent in the finfinite domain can be equivalent in a smaller domain, so more formulas can be rewritten into each fragment of recHML. For example, on the infinite domain the modal operators $\langle a\rangle\psi$ can be rewritten as

$$[a]\psi \wedge \bigwedge_{b\in \text{Act}\setminus\{a\}} [b]\mathsf{ff},$$

and is monitorable for violations whenever ψ is monitorable for violations. In the finfinite domain, this is not the case, since the two formulas evaluate differently on the empty trace. While $\langle a\rangle\psi$ can easily be re-written, some formulas, such as $\mathsf{min}\, X.\bigwedge_{a\in\text{Act}}[a]X$ ("the trace is finite"), which on the infinite domain is equivalent to ff, are not as easily detected. Due to these semantic equivalences, more formulas are monitorable on the infinite domain than in the finfinite domain (see also [5]).

Is it then necessary to analyse monitorability in each domain separately, or can our syntactic characterisations be shown to be robust across domains? In this section we show that sHML and cHML characterise violation and satisfaction monitorability across all linear domains, but that the other syntactic characterisations are not as robust and may require special conditions or variations. Specifically, iHML and pHML, although not robust for arbitrary domains, also characterise informative and persistently informative monitorability for a certain class of domains — and in particular for the finite and the infinite domains. If we parameterize the definition of iHMLwith a semantic condition that depends on the domain, then we can show that it is a robust characterization of informative monitorability across all linear domains.

Monitorability parameterised by domain To begin, we can parameterise soundness, completeness, informativity, and persistent informativity with domains in the obvious way. Soundness in a domain $D \subseteq \text{Act}^\infty$ only requires a monitor to agree with a property over traces in D. Similarly, completeness in D only requires the monitor to identify violations or satisfying traces within D. Informative monitors must reach a verdict with a prefix of a trace in D and persistently informative monitors must be able to do so after reading any prefix of a trace in D. Then, each type of monitorability extends to monitorability in D.

For the following definitions, we fix a domain $D \subseteq \text{Act}^\infty$.

Definition 6.1 (Soundness in domain D) Monitor m is *sound* for property P in domain D if for all $f \in D$:

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

Definition 6.2 (Completeness in domain D) Monitor m is *satisfaction-complete* for P in domain D if $f{\in}P \cap D$ implies $\mathbf{acc}(m, f)$ and *violation-complete* for P if $f{\in}(D \setminus P)$ implies $\mathbf{rej}(m, f)$. It is *complete* for P in D if it is *both* satisfaction- and violation-complete for P in D and *partially-complete* if it is *either* satisfaction- *or* violation-complete in D. ■

Definition 6.3 (Informative Monitors in domain D) A monitor m is:

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

Then, the various grades of monitorability can similarly be parameterized with respect to D in a straightforward manner.

We now show that sHML and cHML are robust descriptions of violation and satisfaction monitorability, in the following sense:

Theorem 6.1 *Given a domain $D \subseteq \text{Act}^\infty$, a property described by a formula $\psi \in$ recHML is monitorable for violations in D if and only if there is a formula $\psi' \in$ sHML such that $[\![\psi]\!] \cap D = [\![\psi']\!] \cap D$.*

Similarly, ψ is monitorable for satisfactions in D if and only if there is a formula $\psi' \in$ cHML such that $[\![\psi]\!] \cap D = [\![\psi']\!] \cap D$.

Proof Since a trace in D is also a trace in Act^∞, a monitor that is sound for ψ' in Act^∞ is also sound for ψ in D. Furthermore, a monitor that is violation-complete for ψ' in Act^∞ is also violation-complete in D, since every violating trace in D is also a violating trace in Act^∞. Hence a sound and violation complete monitor for ψ' in Act^∞ also witnesses the violation-monitorability of ψ in D.

Conversely, let m be a sound and violation-complete monitor in D for ψ. Then, by Theorem 4.3, it is also sound and violation-complete in Act^∞ for some $\psi' \in$ sHML, obtained by formula synthesis from m. Then, $[\![\psi]\!] \cap D \subseteq [\![\psi']\!] \cap D$ since by completeness for ψ', m identifies all traces in $D \setminus [\![\psi']\!]$, and by soundness for ψ, these traces must also be violations of ψ. By a similar argument, $[\![\psi']\!] \cap D \subseteq [\![\psi]\!] \cap D$.

The argument for monitorability for satisfactions is similar. □

In other words, sHML and cHML are the syntactic fragments that describe monitorability for violations and satisfactions *for any* domain $D \subseteq \text{Act}^\infty$, in particular the infinite domain and the finite domain.

This conclusion is in line with the work presented in [5], where we identified sHML and cHML as the fragments describing violation and satisfaction monitorability in the infinite domain. Interestingly, this robustness seems to extend even

to the branching-time domain [38], although this is beyond the scope of the above theorem.

We can then ask whether a similar robustness result holds for the other types of monitorability. In what follows, we show that such a strong result does not hold for any of the other fragments that we have identified so far, yet the fragments for informative and persistently informative monitorability are preserved for the infinite domain Act^ω.

Complete monitorability From its definition, the collection of completely monitorable properties is the intersection of the sets of satisfaction- and violation-monitorable properties. However, this does not mean that this collection is described by the syntactic intersection of sHML and cHML on all domains, even though this is indeed the case in the finfinite domain. Rather, it is captured by a fragment F such that any sHML formula equivalent to some cHML formula over D is also equivalent to a formula of F over D. For example, in the infinite domain, HML, the fragment of recHML without fixpoints, characterises completely monitorable properties [5], and we can see that in certain specific domains, we can completely monitor for all recHML properties. For instance, if the domain is Act^k, the finite traces of length exactly $k \geq 0$, a monitor can simply evaluate any recHML property after observing k events — on the other hand, if the domain allows traces of length *at most* k, then $[\alpha]\mathtt{tt}$ is not monitorable for violations. A purely syntactic characterisation of completely monitorable formulas does not seem plausible, but we can give one with a semantic condition.

Theorem 6.2 *For every* recHML *formula φ and domain $D \subseteq \text{Act}^\infty$, φ is completely monitorable in D if and only if there are formulas $\psi \in$ sHML and $\psi' \in$ cHML, such that*

$$[\![\varphi]\!] \cap D = [\![\psi]\!] \cap D = [\![\psi']\!] \cap D.$$

Proof Theorem 6.2 is a direct consequence of Theorem 6.1. □

Informative and persistently informative monitorability Unlike satisfaction and violation monitorability, informative monitorability, as defined in Definition 3.8, is not robust to variations in the domain. For example, the property "a occurs either immediately or infinitely often" is informatively monitorable in Act^∞ since the finite trace a determines the property. However, if D is pathological, i.e. $D = [\![\neg a]\!]$, then the property is not informatively monitorable in D. Then, there is no hope for the syntactic fragment iHML to be robust in the same way as sHML and cHML are robust.

Similarly, persistently informative monitorability is also not necessarily preserved in subdomains. Indeed, "Either a occurs infinitely often or eventually b" is persistently informative in Act^∞, but not in $(\text{Act} \setminus \{b\})^\infty$.

That being said, the syntactic fragments of recHML that correspond to the informatively and persistently informatively monitorable properties are robust with respect to certain natural domains such as the infinite domain — specifically for the domains whose finite prefixes are Act^*. To show this, we first define the set of prefixes of a domain D to be:

$$pr(D) = \{s \in \text{Act}^* \mid \exists f \cdot sf \in D\}.$$

Lemma 6.1 *Let $D \subseteq \text{ACT}^\infty$, such that $pr(D) = \text{ACT}^*$. Then, for every monitor m, m is informative (resp., persistently informative) monitorable in D if and only if m is informative (resp., persistently informative) monitorable in* ACT^∞.

Theorem 6.3 *Let $D \subseteq \text{ACT}^\infty$, such that $pr(D) = \text{ACT}^*$. Then, for every $\varphi \in$ recHML, φ is informatively (resp., persistently informatively) monitorable in D if and only if there is some $\psi \in$ siHML $\cup$ ciHML (resp., $\psi \in$ pHML), such that $[\![\varphi]\!] \cap D = [\![\psi]\!] \cap D$.*

Proof We first assume that $\varphi \in$ pHML (*resp.*, $\varphi \in$ iHML) and we demonstrate that there is a regular monitor m that is persistently informative (*resp.*, informative) and sound for φ in D. From Theorem 5.4, we see that there is a regular monitor m that is persistently informative (*resp.*, informative) and sound for φ in ACT^∞. From the fact that $pr(D) = \text{ACT}^*$, we get that m is persistently informative (*resp.*, informative) in D (notice that all finite traces can be extended into D), and from the fact that $D \subseteq \text{ACT}^\infty$, we get that m remains sound for φ in D.

For the cases where φ is informatively and persistently informatively monitorable in D, we follow the proofs of Lemma 5.3 and Theorem 5.4 respectively, taking D into account. □

The reason that Theorem 6.3 holds for domains D with $pr(D) = \text{ACT}^*$ is that these domains preserve the finite prefixes that determine a property. The theorem can be generalized to any domain D for the case of siHML $\cup$ ciHML, but after we generalize the definitions of siHML $\cup$ ciHML with the domain D as a parameter, as follows.

Definition 6.4 The informative fragment in D is $\text{iHML}_D = \text{siHML}_D \cup \text{ciHML}_D$ where

$$\text{siHML}_D = \{\varphi_1 \wedge \varphi_2 \in \text{recHML} \mid \varphi_1 \in \text{sHML and } D \setminus [\![\varphi_1]\!] \neq \emptyset\},$$
$$\text{ciHML}_D = \{\varphi_1 \vee \varphi_2 \in \text{recHML} \mid \varphi_1 \in \text{cHML and } D \cap [\![\varphi_1]\!] \neq \emptyset\}.$$ ■

Theorem 6.4 *Let $D \subseteq \text{ACT}^\infty$. Then, for every $\varphi \in$ recHML, φ is informatively monitorable in D if and only if there is some $\psi \in \text{siHML}_D \cup \text{ciHML}_D$, such that $[\![\varphi]\!] \cap D = [\![\psi]\!] \cap D$.*

It is likely that a similar generalization and characterization can be achieved for pHML, but we expect the syntactic condition to be more convoluted.

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

7 Safety and Co-safety

In this section and the next, we study how classic definitions of monitorability correspond to levels of the monitorability hierarchy, starting with safety and co-safety

as definitions of monitorability. While these correspondences between a language-based view of monitorability and an operational view of monitorability might seem straight-forward, there are some subtleties involved. Indeed, in this section, we also discuss an error in Falcone *et al.*'s work that aimed to establish such a correspondence between safety properties and an operational view of monitorability.

The classic (and perhaps the most intuitive) definition of monitorability consists of (some variation of) *safety* properties [5,9,33,40,57,61]. 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 [61] 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 Section 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 [33, Def. 4] and, when restricted to infinite traces, to other work such as [9, 20, 40].

Definition 7.1 (Safety) A property $P \subseteq \text{Act}^\infty$ 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 Figure 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 [33], reachability [19]), is monitorable for satisfaction. That is, every trace that satisfies a co-safety property can be positively determined by a finite prefix.

Definition 7.2 (Co-safety) A property $P \subseteq \text{Act}^\infty$ 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 Figure 1.1. ■

Example 7.1 "Eventually s *is reached", i.e.,* $\mathsf{F\,s}$, is a co-safety property whereas *"*f *never occurs", i.e.,* $\mathsf{G}\neg\mathsf{f}$, is a safety property. The property *"*s *occurs infinitely often", i.e.,* $\mathsf{G\,F\,s}$, is neither safety nor co-safety. The property only holds over infinite traces so it cannot be positively determined by a finite trace. Dually, there is *no* finite trace that determines that there cannot be an infinite number of s occurrences in a continuation of the trace. Similarly, φ_{even} from Example 4.2 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 Figure 1.1.

Theorem 7.1 $\mathsf{VCmp} = \mathsf{Safe}$ *and* $\mathsf{SCmp} = \mathsf{CoSafe}$.

Proof We treat the case for safety, as the case for co-safety is similar. If P is a safety property, then for every $f \in \text{Act}^\infty \setminus P$, there is some finite prefix s of f

that negatively determines P. Therefore, m_P is sound (Lemma 3.3) and violation-complete (Definition 3.2) for P. The other direction follows from the fact that whenever $P \subseteq \text{Act}^\infty$ is monitorable for violation, every $f \in \text{Act}^\infty \setminus P$ 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 Theorem 7.1, we obtain a syntactic characterisation for the Safe and CoSafe monitorability classes; see Figure 1.1.

7.1 The Operational Monitorability Characterisation of Safety due to Falcone *et al.*

Falcone *et al.* [33, Pg. 362, Col. 2] remark that the classical definitions of monitorability are hard to use in practice. They follow a similar agenda to ours, namely that of providing alternative characterisations for the different monitorability classes of properties. They propose three definitions of monitorability [33, Defs. 16 and 17] which they claim to coincide with safety, co-safety, and the union of safety and co-safety properties in [33, Thm. 3]. These alternative definitions elegantly generalise the n-valued logic semantics approach pioneered by Bauer *et al.* for LTL [16, 18]. We view n-valued logic semantics as more operational in flavour because they abstractly describe the expected behaviour of a monitor, in terms to the verdicts it should return when presented with a partial trace. In this sense, these alternative definitions are more amenable for the purposes of correct monitor construction and is in line with our motivations discussed in Section 1.

Obtaining characterisations for monitorability classes can however be quite subtle. Particularly, we here argue that the claim made by [33, Thm. 3] does not hold. We were alerted to this when relating the monitorability hierarchy of Figure 1.1 to the state of the art. Prior to this work, we were of the understanding that the linear-time monitorability classes studied in [5] were different from those studied in [33]. Our judgement was clouded by a number of factors that made the definitions hard to relate, such as the different formalisms used (*e.g.,* recHML vs regular expressions, process calculi vs Strett Automata, *etc.*). However, once were went through the effort of recasting these results in our unifying framework the similarities and discrepancies became easier to spot.

Remark 7.1 Falcone *et al.* present finfinite properties as a pair consisting of a set of finite traces and a set of infinite traces. This difference is cosmetic and here we speak of just one set, containing both finite and infinite traces.

We recall Falcone *et al.*'s definitions and show that their definitions of monitorability include more than just safety and co-safety properties. Their definition of monitorability relies on the notion of a property evaluation parameterised by a truth domain [33, Def. 16]. 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 7.3 (Property evaluation *wrt.* a truth-domain [33, Def. 16]) For each of three different verdict-domains and finfinite properties P ("r-properties" in their terminology), Falcone *et al.* define the following evaluation functions:

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

Definition 7.4 (Alternative FFM-monitorability [33, Def. 17]) A property P is $\mathbb{B}$-monitorable over a truth domain $\mathbb{B}$ if for all $s, r \in \text{Act}^*$, if $s \in P$ and $r \notin P$, then $[\![P]\!]_{\mathbb{B}}(s) \neq [\![P]\!]_{\mathbb{B}}(r)$. ■

In [33, Thm. 3], they then claim that (i) For $\mathbb{B} = \{\mathsf{ff}, ?\}$, $\mathbb{B}$-monitorability equates to Safety (ii) For $\mathbb{B} = \{\mathsf{tt}, ?\}$, $\mathbb{B}$-monitorability equates to Co-safety (iii) For $\mathbb{B} = \{\mathsf{tt}, \mathsf{ff}, ?\}$, $\mathbb{B}$-monitorability equates to Safety $\cup$ Co-safety.

From [33, Def. 17], it easily follows that any property P for which $P \cap \text{Act}^* = \emptyset$ or $\text{Act}^* \subseteq P$ is vacuously $\mathbb{B}$-monitorable for any truth-domain $\mathbb{B}$, and evaluation function. However, not all such properties are necessarily Safety or Co-safety properties which contradicts [33, Thm. 3].

Example 7.2 Recall the property "always eventually s", expressed in LTL as GFs. Using the encoding outlined in Example 4.1, this property can be expressed as the recHML formula:

$$\mathsf{max}\, X.\Big(\mathsf{min}\, Y.\big(\langle \mathsf{s}\rangle \mathsf{tt} \vee \big(\bigvee_{a \in \text{Act}} \langle a\rangle Y\big) \wedge \bigvee_{a \in \text{Act}} \langle a\rangle X\Big)$$

We argued in Example 3.2 that this property is *not* a safety property. From the structure of the recHML formula, one can easily conclude that *no* finite traces satisfy this property.[6] This means that the implication defining Definition 7.4 is trivially satisfied, which in turn means that GFs is $\mathbb{B}$-monitorable. This counter-example contradicts [33, Thm. 3]. ■

We believe the critical error can be traced back to [33, Lem. 3] used by [33, Thm. 3]. In particular, the proof of [33, Lem. 3] (Appendix 2.3) falsely claims that whenever $P \cap \text{Act}^* = \emptyset$ or $\text{Act}^* \subseteq P$, then that property P is necessarily either a safety or co-safety property. This property does not hold even when the domain of discourse is limited to the reactivity class [20], which is obtained by the boolean combination of response and persistence properties.[7]

[6] It can be easily inferred from the outer maximal fixpoint and the exlusive use of existential modalities.

[7] A *response* is one where all of its infinite traces have an infinite number of prefixes satisfying the property as well. A *persistence* property is one where all of its infinite sequences contain a finite number of prefixes that do not satisfy the property.

Example 7.3 Consider the set $P_{\mathsf{f}} = \{\, s\mathsf{f} \mid s \in \textsc{Act}^* \,\}$, *i.e.,* the set of all finite traces ending with a f. We can use the construction in [33, Def. 4] to build from P_{f} the response property P_{f}^{∞} containing all finite traces ending in f and all the infinite traces where f occurs infinitely often. In the same manner, we can build the second response property P_{s}^{∞} containing all finite traces ending in s and all the infinite traces where s occurs infinitely often. Now by the definition of the reactivity class [20], the conjuction $P_{\mathsf{f}}^{\infty} \cap P_{\mathsf{s}}^{\infty}$ should be a reactive property. It should also be clear that $P_{\mathsf{f}}^{\infty} \cap P_{\mathsf{s}}^{\infty}$ does not have any finite traces (*i.e.,* a trace cannot both end with f *and* s) but it clearly contains infinite traces (*e.g.,* $(\mathsf{fs})^{\omega}$). This means that $P_{\mathsf{f}}^{\infty} \cap P_{\mathsf{s}}^{\infty} \neq \mathsf{ff}$, contradicting the key assumption used in the reasoning for the proof of [33, Lem. 3]. ■

This concludes our analysis of how safety and co-safety, used as notions of monitorability, fit into the hierarchy established in Section 3.

8 Pnueli and Zaks

The work on monitorability due to Pnueli and Zaks [53] 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 8.1 ([53]) 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 8.1 The property $\big(\mathsf{f} \wedge \mathsf{F}\,\mathsf{r}\big) \vee \big(\mathsf{F}\,\mathsf{G}\,\mathsf{s}\big)$ is s-monitorable for any finite trace that begins with f, *i.e.,* $\mathsf{f}s$, since it is determined by the extension $\mathsf{f}s\mathsf{r}$. It is *not* s-monitorable for finite traces that begin with an action other than f. ■

Monitorability over properties—rather than over property–sequence pairs—can then be defined by either quantifying *universally* or *existentially* over finite traces: a property is monitorable either if it is s-monitorable for all s, or for some s. We address both definitions, which we call ∀PZ- and ∃PZ-monitorability respectively. ∀PZ-monitorability is the more standard interpretation: it appears for example in [33] where it is attributed to Pnueli and Zaks. However, the original intent seems to align more with ∃PZ-monitorability: in [53], 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 [22]. The earliest definition equivalent to ∀PZ (restricted to infinite words) that we could find is due to Bauer, Leucker and Schallhart [18]. They complement the terminology of Kupferman and Vardi of good and bad prefixes with ugly prefixes, which are those with no good or bad extension. Their definition of monitorability consists of properties without ugly prefixes. It is easy to see that this is in fact exactly equivalent to Pnueli and Zack's definition quantified universally.

Definition 8.2 (∀PZ-monitorability) A property P is (universally Pnueli–Zaks) ∀PZ-monitorable if it is s-monitorable for *all* finite traces s. The class of all ∀PZ-monitorable properties is denoted $\forall\mathsf{PZ}$. ■

Definition 8.3 (∃PZ-monitorability) A property is (existentially Pnueli–Zaks) ∃PZ-monitorable if it is s-monitorable for *some* finite trace s, *i.e.,* if it is ε-monitorable. The class of ∃PZ-monitorable properties is written $\exists\mathsf{PZ}$. ■

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

Example 8.2 Consider the property *"Either* s *occurs before* f*, or* r *happens infinitely often"*, expressed in LTL fashion as $\big((\neg\mathsf{f})\,\mathsf{U}\,\mathsf{s}\big) \vee \big(\mathsf{G}\,\mathsf{F}\,\mathsf{r}\big)$. 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 7.1, 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 Definitions 8.2 and 8.3, it follows immediately that $\forall\mathsf{PZ} \subset \exists\mathsf{PZ}$.

Proposition 8.1 *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 Definition 7.1, 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 Definition 8.1. 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 Figure 1.1.

Theorem 8.1 $\exists\mathit{PZ} = \mathit{ICmp}$ *and* $\forall\mathit{PZ} = \mathit{PICmp}$.

Proof Since the proofs of the two claims are analogous, we simply outline the one for $\forall\mathsf{PZ} = \mathsf{PICmp}$. Let $P \in \forall\mathsf{PZ}$ and pick a finite trace $s \in \text{Act}^*$. By Lemma 3.3, m_P is sound for P. By Definition 3.7 we need to show that there exists an f such that $\mathbf{acc}(m_P, sf)$ or $\mathbf{rej}(m_P, sf)$. From Definitions 8.1 and 8.2 we know that there is a finite r such that sr positively or negatively determines P. By Definition 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 $s \in \text{Act}^*$. By Definitions 8.1 and 8.2, we need to show that there is an extension of s that positively or negatively determines P. From Definitions 3.7 and 3.8, there exists some f such that $\mathbf{acc}(m_P, sf)$ or $\mathbf{rej}(m_P, sf)$. By Definition 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 Definition 3.2, we know that sr either positively or negatively determines P. Thus $P \in \forall\mathsf{PZ}$. □

9 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, and how the framework of the monitorability hierarchy can be adapted to these settings.

Monitorability wrt. the alphabet. The monitorability of a property can depend on Act. For instance, if Act has at least two elements $\{a, b, \ldots\}$, property $\{a^\omega\}$, which can be represented as $\mathsf{max}\ X.\langle a\rangle X$, is s-monitorable for every sequence s, as s can be extended to sb, which negatively determines the property. On the other hand, assume that $\text{Act} = \{a\}$. 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.

Whether the robustness results of Section 6 can be extended from domain variations to alphabet variations is left as future work. It is particularly interesting to consider infinite alphabets, which enable the processing of data. While the definitions of monitorability can easily be extended to infinite alphabets, it is no longer interesting to just consider a maximal monitoring system. Indeed, the ability to parameterise the monitorability hierarchy with different monitoring systems becomes paramount: the choice of monitoring system defines what computational capabilities the monitors are given, for instance whether the monitors can do restricted arithmetic on the data, or just compare it to a fine set of values.

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, Leucker and Schallhart when they define a finite-trace semantics for LTL, called RV-LTL [16]. Falcone *et al.* [33] also have a definition of monitorability based on this idea (in addition to those discussed in Section 7.1). 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.

Other ways to classify RV properties. Havelund and Peled recently presented a related classification of infinitary properties [40]. Their classification is also based on the type of verdicts one can reach from finite traces. Their classification consists of safety and co-safety properties, (there called AFS and AFR), 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). While their classes AFS and AFR coincide with safety and co-safety, their classification is otherwise quite different. Indeed, they show that several of their classes contain both $\forall$Pz-

monitorable and non-∀PZ-monitorable properties. In contrast, in our classification, ∀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. As such, in our hierarchy the core monitorability classes are all either subclasses or superclasses of ∀PZ-monitorability. The Havelund and Peled classification is therefore somewhat orthogonal to ours. However, note that many of their classes can be recovered as boolean combinations of the classes we consider here. For example, NFS is the negation of properties that are informatively monitorable for violations. SFS on the other hand is the set difference of properties informatively monitorable for satisfaction and persistently informatively monitorable for satisfaction.

Another important classification is the safety-progress hierarchy [20]. There the safety and guarantee classes correspond to the safety and co-safety classes discussed earlier. However, the richer classes, that is, response properties, persistence properties, and their boolean combinations (also known as reactive properties), are orthogonal to the monitorability classes considered here.

Monitoring in finite trace domains. 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*; indeed, a monitor can just wait for the end of the trace and then give a verdict. In this setting the question of monitorability is less relevant. However, if the end of the trace is not observable, then this corresponds to restricting the domain to ACT*, and the question of monitorability is again non-trivial, but can be addressed with the techniques developed in Section 6.

Monitorability parameterised by the domain. In Section 6, we observe that instead of considering finite, infinite or 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. Cimatti *et al.* and Leucker consider monitoring with prior knowledge in [23,49]. Leucker proposes an LTL semantics parameterised by this prior knowledge while Cimatti *et al.* incorporate the assumption directly into the monitoring algorithm, thereby treating violations of the assumptions and violations of the property to be monitored differently. Stucki *et al.* parameterise monitorability of hyperproperties with prior knowledge in [59]. The developments in Section 6 provide some insights into how a syntactic approach can be leveraged to approach monitorability with prior knowledge. It is not yet clear, how this approach might extend to hyperproperties.

Monitoring branching-time properties. The domains we consider in this paper, and, in fact, in most bodies of work on monitorability, are linear domains. However, one can also consider the monitorability of branching-time properties, as it was done in [1, 2, 5, 38] for regular properties expressed in RECHML. The version of monitorability the authors of these papers considered is partial monitorability, similarly defined as in Definition 3.6. Specifically, Francalanza *et al.* show that SHML$\cup$CHML is the fragment of RECHML that captures the partially monitorable

fragment of branching time recHML, and Aceto *et al.* [1] extend this to a setting with silent actions and partial obervability. Aceto *et al.* [2] demonstrate how these limits can be extended for the non-classical monitoring setup where the monitors can observe other aspects of the system under scrutiny besides the events executed in the current run; these might include events that could not have been carried out (*i.e.,* refusals [52]) or traces from previous executions of the system. For the classical monitoring setup considered in this article, Aceto *et al.* [5] show how monitorability compares between linear-time and branching-time properties. The paper [37] gives a comprehensive high-level overview of this line of research.

Monitoring non-regular properties. Although we have focussed on the monitorability of regular properties, the monitorability hierarchy of Section 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 in [14], while in [34], Ferrier *et al.* consider monitors with registers (*i.e.,* infinite state monitors) to verify safety properties that are not regular. Bauer, Leucker and Schallhart also consider monitoring timed languages in [18]. In each of these cases, the properties considered span more than just regular languages. Therefore, the syntactic characterisations we provide no longer characterise monitorability for these classes. Characterising (*e.g.,* syntactically) the different classes of monitorability for non-regular properties is left as future work. Note that while context-free languages can directly be understood in the framework of Section 3, this not the case for languages with data or timed languages, which, technically, have an infinite alphabet. However, the alphabet is of little consideration in Section 3 and could, *mutatis mutandis*, be taken to be infinite.

A further dimension of monitorability comes into play with non-regular properties: computability. Indeed, while with regular languages most language manipulations are computable, this is not necessarily the case for more general properties. This means that monitorability has to also take into account the computational power of the monitors and one further has to ask whether the set of good or bad prefixes of a property is computable. Monitorability can therefore be limited by computability. Indeed, Kim and Viswanathan [61] who consider monitorability for properties beyond regular, restrict themselves to recursively enumerable safety properties. More recently, Stucki *et al.* differentiate between theoretical monitorability and computable monitorability of hyperproperties. Interestingly, there is a subtle difference in the approaches here: one can either restrict the whole class of properties to satisfy some computational demands, or just the monitors. This might lead to different hierarchies of monitorability as it is conceivable that, for some properties, their good or bad prefixes might be computationably simpler than the property itself. Indeed, the separation of concerns between the computational objects, the monitors, and the specifications allows the monitorability hierarchy to be parameterised independently by the class of specifications and the class of monitors, enabling an analysis of both approaches.

In our monitorability hierarchy, the set of monitors in the monitoring system handles this dimension: the set of monitors determines the computational power of the monitors, and monitorability is parameterised by this set. For some classes of non-regular properties, a *maximal* monitoring system might not be implementable

or practical and it becomes interesting to consider non-maximal monitoring systems that use limited resources, and the monitorability hierarchies they generate.

Beyond Monitorability. Stream-based monitoring systems such as those presented in [26, 27] 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, 33] is a criterion that is better suited for these settings.

10 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 $\exists\text{PZ}$- and $\forall\text{PZ}$-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 Section 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.

A Appendix: The proof of Theorem 5.3

In this appendix, we present the proof of Theorem 5.3, which was omitted from the main text.

We first define a notion for monitors that is similar to the one of Definition 5.3.

Definition A.1 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 make explicit two straightforward lemmata that we will use.

Lemma A.1 *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 A.2 – *If all subformulas 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 subformulas of φ can refute (or verify).*

– *Let $\varphi = \mathsf{max}\, X.\psi$ or $\varphi = \mathsf{min}\, X.\psi$. If all subformulas of φ can refute (resp., verify), then all subformulas of $\psi[\varphi/X]$ can refute (resp., verify).*

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

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

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

Lemma A.3 *For all possibly open $\varphi, \psi \in$ eHML ∩ sHML, $d_B(\varphi[\psi/X]) \leq d_B(\varphi)$.*

Proof Straightforward induction on φ. □

Lemma A.4 *Let $\alpha \in$ Act.*

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

Proof We assume that $\varphi \in$ eHML ∩ sHML, as the case for $\varphi \in$ eHML ∩ 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 Lemma 5.2.

If $\varphi = \mathsf{ff}$, then we are done immediately by taking $\psi = \mathsf{ff}$.
If $\varphi = \bigwedge_{\gamma\in\text{Act}} [\gamma]\varphi_\gamma$, 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 subformulas.
If $\varphi = \max X.\varphi'$, then $\varphi'[\varphi/X] \in$ eHML∩sHML and all subformulas of $\varphi'[\varphi/X]$ can refute, by Lemma A.2. Furthermore, $[\![\varphi]\!] = [\![\varphi'[\varphi/X]]\!]$, and we are done by the inductive hypothesis. □

Lemma A.5 *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 ∩ sHML and all of its subformulas can refute, and $\psi_* \in$ recHML. By Theorem 4.4, it suffices to prove that for every $s \in \text{Act}^*$, there is some $r \in \text{Act}^*$, such that sr negatively determines φ. We prove this by structural induction on s. If $s = \varepsilon$, then as in the proof of Lemma 5.2, we can show that there is a finite trace that negatively determines ψ. If $s = as'$, then by Lemma A.4. there is some $\psi' \in$ eHML ∩ sHML, such that all subformulas of ψ' can refute, and for every $f \in \text{Act}^\infty$, $af \in [\![\psi]\!]$ 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:

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

Lemma A.6 *Let m be a persistently rejecting, deterministic regular monitor. If $A \subsetneq \text{Act}$, then $\sum_{\alpha\in A} \alpha.m_\alpha$ can only appear in m as a submonitor of a larger sum.*

Proof Let $a \in \text{Act} \setminus A$ 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 $\sum_{\alpha\in A} \alpha.m_\alpha \stackrel{a}{\not\Rightarrow}$. Therefore, it suffices to prove that for every deterministic n with free variable x, if $n' \stackrel{a}{\not\Rightarrow}$, then there is a finite trace s, such that there is no regular monitor o for which $n[n'/x] \stackrel{sa}{\Longrightarrow} o$. 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:

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

Corollary A.1 *Every persistently rejecting, deterministic regular monitor is explicit.*

Proof A consequence of Lemma A.6. □

Lemma A.7 *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 subformulas can refute.*

Proof By induction on the construction of m. □

Lemma A.8 *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 Corollary 4.1, there is a regular monitor, m, that is sound for φ and persistently rejecting. By Theorem 4.1, we can assume that m is deterministic (Definition 4.2). From Corollary A.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 $m \stackrel{sr}{\Longrightarrow} \mathsf{no}$, which is a contradiction. Observe that $\mathsf{f}(m) \in$ sHML. Then, from Lemma A.7, the sHML formula $\mathsf{f}(m)$ is in eHML, and all of its subformulas 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 *Let $\varphi \in$ recHML. Then, φ is persistently informatively monitorable for violation if and only if there is some $\psi \in$ spHML, such that $[\![\psi]\!] = [\![\varphi]\!]$; φ is persistently informatively monitorable for satisfaction if and only if there is some $\psi \in$ cpHML, such that $[\![\psi]\!] = [\![\varphi]\!]$.*

Proof A consequence of Lemmata A.5 and A.8.

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. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology **20**(4), 14 (2011)
19. 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)
20. 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
21. 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

22. 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)
23. Cimatti, A., Tian, C., Tonetta, S.: Assumption-based runtime verification with partial observability and resets. In: International Conference on Runtime Verification, pp. 165–184. Springer (2019)
24. 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
25. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT press (1999)
26. 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
27. 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)
28. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B.: Uppaal SMC tutorial. Int. J. Softw. Tools Technol. Transf. **17**(4), 397–415 (2015). DOI 10.1007/s10009-014-0361-y. URL https://doi.org/10.1007/s10009-014-0361-y
29. 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
30. Decker, N., Leucker, M., Thoma, D.: jUnit$^{\mathrm{rv}}$-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
31. 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)
32. 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
33. 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)
34. 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
35. 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)
36. 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
37. 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
38. 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
39. 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
40. 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
41. Hennessy, M., Milner, R.: Algebraic Laws for Nondeterminism and Concurrency. Journal of the ACM **32**(1), 137–161 (1985). DOI 10.1145/2455.2460

42. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation. Acm Sigact News **32**(1), 60–65 (2001)
43. Kozen, D.C.: Results on the propositional μ-calculus. Theoretical Computer Science **27**, 333–354 (1983)
44. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Formal Methods in System Design **19**(3), 291–314 (2001)
45. 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)
46. 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
47. Larsen, K.G., Lorber, F., Nielsen, B.: 20 years of UPPAAL enabled industrial model-based validation and beyond. In: T. Margaria, B. Steffen (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part IV, *Lecture Notes in Computer Science*, vol. 11247, pp. 212–229. Springer (2018). DOI 10.1007/978-3-030-03427-6_18. URL `https://doi.org/10.1007/978-3-030-03427-6_18`
48. Laurent, J., Goodloe, A., Pike, L.: Assuring the Guardians. In: Runtime Verification (RV), *LNCS*, vol. 9333, pp. 87–101 (2015)
49. Leucker, M.: Sliding between model checking and runtime verification. In: International Conference on Runtime Verification, pp. 82–87. Springer (2012)
50. Manna, Z., Pnueli, A.: Completing the temporal picture. Theoretical Computer Science **83**(1), 97–130 (1991). DOI 10.1016/0304-3975(91)90041-Y
51. 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`
52. Phillips, I.: Refusal testing. Theor. Comput. Sci. **50**, 241–284 (1987). DOI 10.1016/0304-3975(87)90117-4. URL `https://doi.org/10.1016/0304-3975(87)90117-4`
53. 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`
54. 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`
55. Rosu, G.: On safety properties and their monitoring. Scientific Annals of Computer Science **22**(2), 327–365 (2012)
56. Sánchez, C., Leucker, M.: Regular linear temporal logic with past. In: G. Barthe, M. Hermenegildo (eds.) Verification, Model Checking, and Abstract Interpretation, pp. 295–311. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
57. Schneider, F.B.: Enforceable security policies. ACM Transactions on Information and System Security **3**(1), 30–50 (2000)
58. Steffen, B., Ingólfsdóttir, A.: Characteristic formulae for processes with divergence. Inf. Comput. **110**(1), 149–163 (1994). DOI 10.1006/inco.1994.1028. URL `https://doi.org/10.1006/inco.1994.1028`
59. 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)
60. 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`
61. 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`
62. 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`