A Framework for Parameterized Monitorability*

Luca Aceto(0000-0002-2197-3018)[1,2], Antonis Achilleos(0000-0002-1314-333X)[2], Adrian Francalanza(0000-0003-3829-7391)[3], and Anna Ingólfsdóttir(0000-0001-8362-3075)[2]

[1] Gran Sasso Science Institute, L'Aquila, Italy
[2] School of Computer Science, Reykjavik University, Reykjavik, Iceland
{luca,antonios,annai}@ru.is
[3] Dept. of Computer Science, ICT, University of Malta, Msida, Malta
adrian.francalanza@um.edu.mt

Abstract. We introduce a general framework for Runtime Verification, parameterized with respect to a set of conditions. These conditions are encoded in the trace generated by a monitored process, which a monitor can observe. We present this parameterized framework in its general form and prove that it corresponds to a fragment of HML with recursion, extended with these conditions. We then show how this framework can be applied to a number of instantiations of the set of conditions.

1 Introduction

Runtime Verification (RV) is a lightweight verification technique that checks whether a system satisfies a correctness property by analysing the *current execution* of the system [20, 29], expressed as a trace of execution events. Using the additional information obtained at runtime, the technique can often mitigate state explosion problems typically associated with more traditional verification techniques. At the same time, limiting the verification analysis to the current execution trace hinders the expressiveness of RV when compared to more exhaustive approaches. In fact, there are correctness properties that cannot be satisfactorily verified at runtime (*e.g.* the finiteness of the trace considered up to the current execution point prohibits the verification of liveness properties). Because of this reason, RV is often used as part of a multi-pronged approach towards ensuring system correctness [5,6,8,14,15,25], *complementing* other verification techniques such as model checking, testing and type checking.

In order to attain an effective verification strategy consisting of multiple verification techniques that include RV, it is crucial to understand the expressive power of each technique: one can then determine how to best decompose the verification burden into subtasks that can then be assigned to the most appropriate verification technique. *Monitorability* concerns itself with identifying the properties that are analysable by RV. In [21, 22] (and subsequently in [2]), the problem of monitorability was studied for properties expressed in a variant of the

* This research was supported by the project "TheoFoMon: Theoretical Foundations for Monitorability" (grant number: 163406-051) of the Icelandic Research Fund.

modal μ-calculus [26] called μHML [28]. The choice of the logic was motivated by the fact that it can embed widely used logics such as CTL and LTL, and by the fact that it is agnostic of the underlying verification method used—this leads to better separation of concerns and guarantees a good level of generality for the results obtained. The main result in [2, 21, 22] is the identification of a monitorable syntactic subset of the logic μHML (*i.e.,* a set of logical formulas for which monitors carrying out the necessary runtime analysis exist) that is shown to be maximally expressive (*i.e.,* any property that is monitorable in the logic may be expressed in terms of this syntactic subset). We are unaware of other maximality results of this kind in the context of RV.

In this work we strive towards extending the monitorability limits identified in [2, 21, 22] for μHML. Particularly, for any logic or specification language, monitorability is a function of the underlying monitoring setup. In [2, 21, 22], the framework assumes a *classical* monitoring setup, whereby a (single) monitor incrementally analyses an ordered trace of events describing the computation steps that were executed by the system. A key observation made by this paper is that, in general, execution traces need *not* be limited to the reporting of events *that happened*. For instance, they may describe events that *could not have happened* at specific points in the execution of a system. Alternatively, they may also include descriptions for depth-bounded trees of computations that *were possible* at specific points in an execution. We conjecture that there are instances where this additional information can be feasibly encoded in a trace, either dynamically or by way of a pre-processing phase (based, *e.g.*, on the examination of logs of previous system executions, or on the full static checking of sub-components making up the system). More importantly, this additional information could, in principle, permit the verification of more properties at runtime.

The contribution of this paper is a study of how the aforementioned augmented monitoring setups may affect the monitorability of μHML, potentially extending the maximality limits identified in [2, 21, 22]. More concretely:

1. We show how these aspects can be expressed and studied in a general monitoring framework with (abstract) conditions, Theorem 3 and Theorem 4 *resp.* in Section 3 and Section 5.
2. We instantiate the general framework with trace conditions that describe the inability to perform actions, amounting to refusals [31], Proposition 1 and Proposition 5.
3. We also instantiate the framework with conditions describing finite execution graphs, amounting to the recursion-free fragment of the logic [24], Proposition 2 and Proposition 3.
4. Finally, we instantiate the framework with trace conditions that record information from previous monitored runs of the system, Proposition 4. This, in turn, leads us to a notion of alternating monitoring that allows monitors to aggregate information over monitored runs. We show that this extends the monitorable fragment of our logic in a natural and significant way.

The remainder of the paper is structured as follows. After outlining the necessary preliminaries in Section 2, we develop our parameterized monitoring framework with conditions in Section 3 for a monitoring setup that allows monitors to observe both silent and external actions of systems. The two condition instantiations for this strong setting are presented in Section 4. In Section 5 we extend the parameterized monitoring framework with conditions to a weak monitoring setup that abstracts from internal moves, followed by two instantiations similar to those presented in Section 4. Section 6 concludes by discussing related and future work.

2 Background

Labelled Transition Systems. We assume a set of *external* actions $\textsc{Act}$ and a distinguished *silent* action τ. We let α range over $\textsc{Act}$ and μ over $\textsc{Act} \cup \{\tau\}$. A *Labelled Transition System* (LTS) on $\textsc{Act}$ is a triple

$$L = \langle P, \textsc{Act}, \rightarrow_L \rangle,$$

where P is a nonempty set of system states referred to as *processes* $p, q, \ldots$, and $\rightarrow_L \subseteq P \times (\textsc{Act} \cup \{\tau\}) \times P$ is a transition relation. We write $p \xrightarrow{\mu}_L q$ instead of $(p, \mu, q) \in \rightarrow_L$. By $p \xrightarrow{\mu}_L$ we mean that there is some q such that $p \xrightarrow{\mu}_L q$. We use $p \overset{\mu}{\Rightarrow}_L q$ to mean that, in L, p can derive q using a single μ action and any number of silent actions, *i.e.,* $p(\xrightarrow{\tau}_L)^* \xrightarrow{\mu}_L (\xrightarrow{\tau}_L)^* q$. We distinguish between (general) traces $s = \mu_1\mu_2 \ldots \mu_r \in (\textsc{Act} \cup \{\tau\})^*$ and *external traces* $t = \alpha_1\alpha_2 \ldots \alpha_r \in \textsc{Act}^*$. For a general trace $s = \mu_1\mu_2 \ldots \mu_r \in (\textsc{Act} \cup \{\tau\})^*$, $p \xrightarrow{s}_L q$ means $p \xrightarrow{\mu_1}_L \xrightarrow{\mu_2}_L \ldots \xrightarrow{\mu_r}_L q$; and for an external trace $t = \alpha_1\alpha_2 \ldots \alpha_r \in \textsc{Act}^*$, $p \overset{t}{\Rightarrow}_L q$ means $p \overset{\alpha_1}{\Longrightarrow}_L \overset{\alpha_2}{\Longrightarrow}_L \ldots \overset{\alpha_r}{\Longrightarrow}_L q$ when $r \geq 1$ and $p(\xrightarrow{\tau})^* q$ when $t = \varepsilon$ is the empty trace. We occasionally omit the subscript L when it is clear from the context.

Example 1. The (standard) regular fragment of CCS [30] with grammar:

$$p, q \in \textsc{Proc} ::= \ \mathsf{nil} \quad | \ \mu.p \quad | \ p + q \quad | \ \mathsf{rec}\ x.p \quad | \ x,$$

where $x, y, z, \ldots$ are from some countably infinite set of variables $\textsc{Var}$, and the transition relation defined as:

$$\textsc{Act}\ \frac{}{\mu.p \xrightarrow{\mu} p} \qquad \textsc{Rec}\ \frac{p[\mathsf{rec}\ x.p/x] \xrightarrow{\mu} q}{\mathtt{rec}x.p \xrightarrow{\mu} q} \qquad \textsc{SelL}\ \frac{p \xrightarrow{\mu} p'}{p + q \xrightarrow{\mu} p'} \qquad \textsc{SelR}\ \frac{q \xrightarrow{\mu} q'}{p + q \xrightarrow{\mu} q'}$$

constitutes the LTS $\langle \textsc{Proc}, \textsc{Act}, \rightarrow \rangle$. We often use the CCS notation above to describe processes. ■

Specification Logic. Properties about the behaviour of processes may be specified via the logic μHML [4, 28], a reformulation of the modal μ-calculus [26].

Definition 1. *μHML formulae on* Act *are defined by the grammar:*

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

where $X, Y, Z, \ldots$ come from a countably infinite set of logical variables LVar*. For a given LTS $L = \langle P, \text{Act}, \rightarrow\rangle$, an environment ρ is a function $\rho : \text{LVar} \rightarrow 2^P$. Given an environment ρ, $X \in \text{LVar}$, and $S \subseteq P$, $\rho[x \mapsto S]$ denotes the environment where $\rho[X \mapsto S](X) = S$ and $\rho[X \mapsto S](Y) = \rho(Y)$, for all $Y \neq X$. The semantics of a μHML formula φ over an LTS L relative to an environment ρ, denoted as $[\![\varphi, \rho]\!]_L$, is defined as follows:*

$$[\![\mathit{tt}, \rho]\!]_L = P \qquad [\![\mathit{ff}, \rho]\!]_L = \emptyset \qquad [\![X, \rho]\!]_L = \rho(X)$$

$$[\![\varphi_1 \wedge \varphi_2, \rho]\!]_L = [\![\varphi_1, \rho]\!]_L \cap [\![\varphi_2, \rho]\!]_L \qquad [\![\varphi_1 \vee \varphi_2, \rho]\!]_L = [\![\varphi_1, \rho]\!]_L \cup [\![\varphi_2, \rho]\!]_L$$

$$[\![[\mu]\varphi, \rho]\!]_L = \left\{ p \mid \forall q.\ p \xrightarrow{\mu} q \text{ implies } q \in [\![\varphi, \rho]\!]_L \right\}$$

$$[\![\langle\mu\rangle\varphi, \rho]\!]_L = \left\{ p \mid \exists q.\ p \xrightarrow{\mu} q \text{ and } q \in [\![\varphi, \rho]\!]_L \right\}$$

$$[\![\mathit{min}\ X.\varphi, \rho]\!]_L = \bigcap \{ S \mid S \supseteq [\![\varphi, \rho[X \mapsto S]]\!]_L \}$$

$$[\![\mathit{max}\ X.\varphi, \rho]\!]_L = \bigcup \{ S \mid S \subseteq [\![\varphi, \rho[X \mapsto S]]\!]_L \}$$

Formulas φ and ψ are equivalent, denoted as $\varphi \equiv \psi$, when $[\![\varphi, \rho]\!]_L = [\![\psi, \rho]\!]_L$ for every environment ρ and LTS L. We often consider closed formulae and simply write $[\![\varphi]\!]_L$ for $[\![\varphi, \rho]\!]_L$ when the semantics of φ is independent of ρ. ■

The logic μHML is very expressive. It is also agnostic of the technique to be employed for verification. The property of monitorability, however, fundamentally relies on the monitoring setup considered.

Monitoring Systems. A *monitoring setup* on Act is a triple $\langle M, I, L\rangle$, where L is a system LTS on Act, M is a monitor LTS on Act, and I is the instrumentation describing how to compose L and M into an LTS, denoted by $I(M, L)$, on Act. We call the pair (M, I) a *monitoring system* on Act. For $M = \langle \text{Mon}, \text{Act}, \rightarrow_M\rangle$, Mon is set of monitor states (ranged over by m) and $\rightarrow_M$ is the *monitor semantics* described in terms of the behavioural state transitions a monitor takes when it analyses trace events $\mu \in \text{Act} \cup \{\tau\}$. The states of the composite LTS $I(M, L)$ are written as $m \triangleleft p$, where m is a monitor state and p is a system state; the monitored-system transition relation is denoted here by $\rightarrow_{I(M,L)}$. We present our results with a focus on *rejection* monitors, *i.e.,* monitors with a designated rejection state no, and hence safety fragments of the logic μHML. However, our results and arguments apply dually to acceptance monitors (with a designated acceptance state yes) and co-safety properties; see [21,22] for details.

Definition 2. *Fix a monitoring setup $\langle M, I, L\rangle$ on* Act *and let m be a monitor state of M and φ a closed formula of μHML on* Act*. We say that m*

(M,I)-rejects *(or simply* rejects*, if M, I are evident) a process p in L, written as* $\mathbf{rej}_{\langle M,I,L\rangle}(m,p)$*, when there are a process q in L and a trace $s \in (\text{Act} \cup \{\tau\})^*$ such that $m \triangleleft p \xrightarrow{s}_{I(M,L)} \mathsf{no} \triangleleft q$. We say that m (M,I)-*monitors for φ on L whenever*

$$\textit{for each process } p \textit{ of } L,\ \mathbf{rej}_{\langle M,I,L\rangle}(m,p) \textit{ if and only if } p \notin [\![\varphi]\!]_L.$$

*(Subscripts are omitted when they are clear from the context.) Finally, m (M,I)-*monitors for φ *when m (M,I)-monitors for φ on L for every LTS L on* Act*. The monitoring system (M,I) is often omitted when evident.* ■

We define monitorability for μHML in terms of monitoring systems (M,I).

Definition 3. *Fix a monitoring system (M,I) and a fragment Λ of μHML. We say that (M,I) rejection-monitors for Λ whenever:*

- *For all closed $\varphi \in \Lambda$, there exists an m from M that (M,I)-monitors for φ.*
- *For all m of M, there exists a closed $\varphi \in \Lambda$ that is (M,I)-monitored by m.* ■

We note that if a monitoring system and a fragment Λ of μHML satisfy the conditions of Definition 3, then Λ is the largest fragment of μHML that is monitored by the monitoring system. Stated otherwise, any other logic fragment Λ' that satisfies the conditions of Definition 3 must be equally expressive to Λ, *i.e.,* $\forall \varphi' \in \Lambda' \cdot \exists \varphi \in \Lambda \cdot \varphi \equiv \varphi'$ and vice versa. Definition 3 can be dually given for acceptance-monitorability, when considering acceptance monitors. We next review two monitoring systems that respectively rejection-monitor for two different fragments of μHML. We omit the corresponding monitoring systems for acceptance-monitors, that monitor for the dual fragments of μHML.

The Basic Monitoring Setup. The following monitoring system, presented in [2], does *not* distinguish between silent actions and external actions.

Definition 4. *A* basic *monitor on* Act *is defined by the grammar:*

$$m, n \in \text{Mon}_b ::= \ \mathsf{end} \ \mid \ \mathsf{no} \ \mid \ \mu.m \ \mid \ m + n \ \mid \ \mathsf{rec}\ x.m \ \mid \ x,$$

where x comes from a countably infinite set of monitor variables. Constant no *denotes the* rejection verdict *state whereas* end *denotes the* inconclusive verdict *state. The basic monitor LTS M_b is the one whose states are the closed monitors of* Mon_b *and whose transition relation is defined by the (standard) rules in Table 1 (we elide the symmetric rule for $m + n$).* ■

Note that by rule mVrd in Table 1, verdicts are irrevocable and monitors can only describe suffix-closed behaviour.

Definition 5. *Given a system LTS L and a monitor LTS M that agree on* Act*, the* basic *instrumentation LTS, denoted by $I_b(M,L)$, is defined by the rules* iMon *and* iTer *in Table 1. (We do* not *consider rule* iAbs *for now.)* ■

Monitor Semantics

$$\textsc{mRec}\ \frac{m[\mathsf{rec}\ x.m/x] \xrightarrow{\mu} m'}{\mathsf{rec}\ x.m \xrightarrow{\mu} m'} \qquad \textsc{mSel}\ \frac{m \xrightarrow{\mu} m'}{m + n \xrightarrow{\mu} m'} \qquad \textsc{mAct}\ \frac{}{\mu.m \xrightarrow{\mu} m} \qquad \textsc{mVrd}\ \frac{}{v \xrightarrow{\mu} v}$$

Instrumentation Semantics

$$\textsc{iMon}\ \frac{p \xrightarrow{\mu}_L q \quad m \xrightarrow{\mu}_M n}{m \triangleleft p \xrightarrow{\mu}_{I(M,L)} n \triangleleft q} \qquad \textsc{iTer}\ \frac{p \xrightarrow{\mu}_L q \quad m \overset{\mu}{\not\longrightarrow}_M}{m \triangleleft p \xrightarrow{\mu}_{I(M,L)} \mathsf{end} \triangleleft q} \qquad \textsc{iAbs}\ \frac{p \xrightarrow{\tau}_L q}{m \triangleleft p \xrightarrow{\tau}_{I(M,L)} m \triangleleft q}$$

Table 1. Behaviour and Instrumentation Rules for Monitored Systems ($v \in \{\mathsf{end}, \mathsf{no}\}$).

Instrumentation often relegates monitors to a passive role, whereby a monitored system transitions only when the system itself can. In rule iMon, when the system produces a trace event μ that the monitor is able to analyse (and transition from m to n), the constituent components of a monitored system $m \triangleleft p$ move in lockstep. Conversely, when the system produces an event μ that the monitor is *unable* to analyse, the monitored system still executes, according to iTer, but the monitor transitions to the inconclusive state, where it remains for the rest of the computation.

We refer to the pair (M_b, I_b) from Definition 4 and Definition 5 as the *basic monitoring system*. For each system LTS L that agrees with the full monitoring system on Act, we can show a correspondence between the respective monitoring setup $\langle M_b, I_b, L\rangle$ and the following syntactic subset of μHML.

Definition 6. *The* safety μHML *is defined by the grammar:*

$$\theta, \chi \in \text{sHML} ::= \ \mathit{tt} \ \mid \ \mathit{ff} \ \mid \ [\mu]\theta \ \mid \ \theta \wedge \chi \ \mid \ \mathit{max}\ X.\theta \ \mid \ X \quad \blacksquare$$

Theorem 1 ([2]). *The* basic *monitoring system* (M_b, I_b) *monitors for the logical fragment* sHML. □

The proof of Theorem 1 relies on a monitor synthesis and a formula synthesis function. The monitor synthesis function, $(\!|-|\!) : \text{sHML} \to \text{Mon}_b$, is defined on the structure of the input formula and assumes a bijective mapping between formula variables and monitor recursion variables:

$$(\!|\mathsf{tt}|\!) = \mathsf{end} \qquad (\!|\mathsf{ff}|\!) = \mathsf{no} \qquad (\!|X|\!) = x$$

$$(\!|[\mu]\psi|\!) = \begin{cases} \mathsf{end} & \text{if } (\!|\psi|\!) = \mathsf{end} \\ \mu.(\!|\psi|\!) & \text{otherwise} \end{cases} \qquad (\!|\mathsf{max}\ X.\psi|\!) = \begin{cases} \mathsf{end} & \text{if } (\!|\psi|\!) = \mathsf{end} \\ \mathsf{rec}\ x.(\!|\psi|\!) & \text{otherwise} \end{cases}$$

$$(\!|\psi_1 \wedge \psi_2|\!) = \begin{cases} (\!|\psi_1|\!) & \text{if } (\!|\psi_2|\!) = \mathsf{end} \\ (\!|\psi_2|\!) & \text{if } (\!|\psi_1|\!) = \mathsf{end} \\ (\!|\psi_1|\!) + (\!|\psi_2|\!) & \text{otherwise} \end{cases}$$

The case analyses in the above synthesis procedure handle some of the redundancies that may be present in formula specifications. For instance, it turns out that $\mathsf{max}\ X.[\mu]\mathsf{tt} \equiv \mathsf{tt}$ and, accordingly, $(\!|\mathsf{max}\ X.[\mu]\mathsf{tt}|\!) = (\!|\mathsf{tt}|\!) = \mathsf{end}$. The formula synthesis function is defined analogously (see [22] and [2] for more details).

Monitoring for External Actions. The results obtained in [21, 22] can be expressed and recovered within our more general framework. We can express a weak version of the modalities employed in [3, 21, 22] as follows:

$$[[\mu]]\varphi \equiv \mathsf{max}\ X.([\tau]X \wedge [\mu]\mathsf{max}\ Y.(\varphi \wedge [\tau]Y)) \quad \text{and}$$
$$\langle\langle\mu\rangle\rangle\varphi \equiv \mathsf{min}\ X.(\langle\tau\rangle X \vee \langle\mu\rangle\mathsf{min}\ Y.(\varphi \vee \langle\tau\rangle Y)).$$

Definition 7. Weak safety μHML*, presented in [21,22], is defined by the grammar:*

$$\pi, \kappa \in \text{WsHML} ::= \ \mathit{tt} \ \mid \mathit{ff} \ \mid [[\alpha]]\pi \ \mid \pi \wedge \kappa \ \mid \mathit{max}\ X.\pi \ \mid X. \quad \blacksquare$$

Definition 8. *The set* Mon_e *of external monitors on* Act *contains all the basic monitors that do not use the silent action* τ*. The corresponding external monitor LTS* M_e*, is defined similarly to* M_b*, but with the closed monitors in* Mon_e *as its states.* External instrumentation*, denoted by* I_e*, is defined by the* three *rules* iMon*,* iTer *and* iAbs *in Table 1, where in the case of* iMon *and* iTer*, action* μ *is substituted by the external action* α*. We refer to the pair* (M_e, I_e) *as the* external monitoring system*, amounting to the setup in [21, 22].* $\blacksquare$

Theorem 2 ([22]). *The* external *monitoring system* (M_e, I_e) *rejection-monitors for the logical fragment* WsHML*.* $\square$

3 Monitors that Detect Conditions

Given a set of processes P, a pair (C, r) is a condition framework when C is a non-empty set of *conditions* and $r : C \to 2^P$ is a valuation function. We assume a fixed condition framework (C, r) and we extend the syntax and semantics of μHML so that for every condition $c \in C$, both c and $\neg c$ are formulas and for every LTS L on set of processes P, $[\![c]\!] = r(c)$ and $[\![\neg c]\!] = P \setminus r(c)$. We call the extended logic $\mu\text{HML}^{(C,r)}$. Since, in all the instances we consider, r is easily inferred from C, it is often omitted and we simply write C instead of (C, r) and $\mu\text{HML}^{(C,r)}$ as μHML^C. We say that process p satisfies c when $p \in [\![c]\!]$. We assume that C is closed under negation, meaning that for every $c \in C$, there is some $c' \in C$, such that $[\![c']\!] = [\![\neg c]\!]$. Conditions represent certain properties of processes that the instrumentation is able to report.

We extend the syntax of monitors, so that if m is a monitor and c a condition, then $c.m$ is a monitor. The idea is that if $c.m$ detects that the process satisfies c, then it can transition to m.

Definition 9. *A* basic C-monitor *on* Act *is defined by the grammar:*

$$m, n \in \text{Mon}_b^C ::= \ \mathit{end} \ \mid \mathit{no} \ \mid \mu.m \ \mid c.m \ \mid m + n \ \mid \mathit{rec}\ x.m \ \mid x,$$

where x *comes from a countably infinite set of monitor variables and* $c \in C$*. Basic* C*-monitor behaviour is defined as in Table 1, but allowing* μ *to range over* $\text{Act} \cup C \cup \{\tau\}$*. We call the resulting monitor LTS* M_b^C*.* $\blacksquare$

A monitor detects the satisfaction of condition c when the monitored system has transitioned to a process that satisfies c. To express this intuition, we add rule iCon to the instrumentation rules of Table 1:

$$\textsc{iCon}\ \frac{p \in [\![c]\!] \ \text{ and } \ m \xrightarrow{c}_M n}{m \triangleleft p \ \xrightarrow{\tau}_{I(M,L)} \ n \triangleleft p}.$$

We call the resulting instrumentation I_b^C. We observe that the resulting monitor setup is transparent with respect to external actions: an external trace of the monitored system results in exactly the same external trace of the instrumentation LTS. However, the general traces are not preserved, as the rule iCon may introduce additional silent transitions for the instrumentation trace. However, we argue that this is an expected consequence of the instrumentation verifying the conditions of C. C-monitors monitor for sHML^C:

Definition 10. *The strong safety fragment of* μHML^C *is defined as:*

$$\varphi, \psi \in \text{sHML}^C ::= \ \mathit{tt} \ \mid \ \mathit{ff} \ \mid \ [\mu]\varphi \ \mid \ \neg c \vee \varphi \ \mid \ \varphi \wedge \psi \ \mid \ \mathit{max}\ X.\varphi \ \mid \ X,$$

where $c \in C$. *We note that* $\neg c \vee \varphi$ *can be viewed as an implication* $c \rightarrow \varphi$ *asserting that if* c *holds, then* φ *must also hold.* ■

It is immediate to see that sHML^C is a fragment of μHML^C and when $C \subseteq \mu\text{HML}$, it is also a fragment of μHML. Finally, if C is closed under negation, then $\neg c \vee \varphi$ can be rewritten as $c' \vee \varphi$, where $[\![c']\!] = [\![\neg c]\!]$, and in the following we often take advantage of this equivalence to simplify the syntax of sHML^C.

Theorem 3. *The monitoring system* (M_b^C, I_b^C) *monitors for* sHML^C. □

We note that Theorem 3 implies that sHML^C is the largest monitorable fragment of μHML^C, relative to C.

4 Instantiations

We consider two possible instantiations for parameter C in the framework presented in Section 3. Since each of these instantiations consists of a fragment from the logic μHML itself, they both show how monitorability for μHML can be extended when using certain augmented traces.

4.1 The Inability to Perform an Action

The monitoring framework of [2, 22] (used also in other works such as [18, 19]), is based on the idea that, while a system is executing, it performs discrete computational steps called events (actions) that are recorded and relayed to the monitor for analysis. Based on the analysed events, the monitor then transitions from state to state. One may however also consider instrumentations that record a system's *inability* to perform a certain action. Examples of this arise

naturally in situations where actions are requested unsuccessfully by an external entity on a system, or whenever the instrumentation is able to report system stability (*i.e.,* the inability of performing internal actions). For instance, such observations were considered in [1, 31], in the context of testing preorders.

In our setting, a process is unable to perform action μ exactly when it satisfies $[\mu]\mathsf{ff}$. For monitors that are able to detect the inability or failure of a process to perform actions, we set $F_{\text{Act}} = \{[\mu]\mathsf{ff} \mid \mu \in \text{Act} \cup \{\tau\}\}$ as the set of conditions. By Theorem 3, the resulting maximal monitorable fragment of μHML is given by the grammar:

$$\varphi, \psi \in \text{sHML}^{F_{\text{Act}}} ::= \mathsf{tt} \mid \mathsf{ff} \mid [\mu]\varphi \mid \langle\mu\rangle\mathsf{tt} \vee \varphi \mid \varphi \wedge \psi \mid \mathsf{max}\ X.\varphi \mid X.$$

We note the fact that μHML is closed under negation, where $\neg[\mu]\mathsf{ff} = \langle\mu\rangle\mathsf{tt}$.

Proposition 1. *The monitoring system* $(M_b^{F_{\text{Act}}}, I_b^{F_{\text{Act}}})$ *monitors for the logical fragment* $\text{sHML}^{F_{\text{Act}}}$. □

A special case of interest are monitors that can detect process stability, *i.e.,* processes satisfying $[\tau]\mathsf{ff}$. Such monitors monitor for $\text{sHML}^{\{[\tau]\mathsf{ff}\}}$, namely sHML from Definition 6 extended with formulas of the form $\langle\tau\rangle\mathsf{tt} \vee \varphi$.

4.2 Depth-Bounded Static Analysis

In multi-pronged approaches using a combination of verification techniques, one could statically verify parts of a program (from specific execution points) with respect to certain behavioural properties using techniques such as Bounded Model Checking [11] and Partial Model Checking [7]. Typical examples arise in component-based software using modules, objects or agents that can be verified in isolation. This pre-computed verification can then be recorded as annotations to a component and subsequently reported by the instrumentation as part of the execution trace. This strategy would certainly be feasible for depth-bounded static analysis for which the original logic HML [24]—the recursion-free fragment of μHML given below—is an ideal fit.

$$\eta, \chi \in \text{HML} ::= \mathsf{tt} \mid \mathsf{ff} \mid \eta \wedge \chi \mid \eta \vee \chi \mid [\mu]\eta \mid \langle\mu\rangle\eta.$$

Again, HML is closed under negation [4]. If we allow monitors to detect the satisfaction of these kinds of conditions, then, according to Theorem 3, the maximal fragment of μHML that we can monitor for, with HML as a condition framework, is sHML^{HML}, defined by the following grammar:

$$\varphi, \psi ::= \mathsf{tt} \mid \mathsf{ff} \mid [\mu]\varphi \mid \eta \vee \varphi \mid \varphi \wedge \psi \mid \mathsf{max}\ X.\varphi \mid X,$$

where $\eta \in \text{HML}$. Another way to describe sHML^{HML} is as the μHML fragment that includes all formulas whereby for every subformula of the form $\varphi \vee \psi$, at most *one* of the constituent subformulas φ, ψ uses recursion.

Proposition 2. *The monitoring system* $(M_b^{\mathrm{HML}}, I_b^{\mathrm{HML}})$ *monitors for the logical fragment* $\mathrm{sHML}^{\mathrm{HML}}$. □

Instead of HML, we can alternatively use a fragment HML^d of HML that only allows formulas with nesting depth for the modalities of at most d. Since the complexity of checking HML formulas is directly dependent on this modal depth, there are cases where the overheads of checking such formulas are deemed to be low enough to be adequately checked for at runtime instead of checking for them statically.

5 Extending External Monitorability

We explore the impact of considering traces that encode conditions from Section 3 on the monitorability of the weak version of the logic used in [21, 22]:

$$\begin{array}{rcl} \varphi, \psi \in \mathrm{W}\mu\mathrm{HML} ::= & \mathtt{tt} \mid \mathtt{ff} \mid \varphi \wedge \psi \mid \varphi \vee \psi & \\ & \mid \langle\langle\alpha\rangle\rangle\varphi \mid [[\alpha]]\varphi \mid \mathsf{min}\ X.\varphi \mid \mathsf{max}\ X.\varphi \mid X. & \end{array}$$

This version of the logic abstracts away from internal moves performed by the system—note that the weak modality formulas are restricted to external actions α as opposed to the general ones, μ. The semantics follows that presented in Section 2, but can alternatively be given a more direct inductive definition, *e.g.*

$$[\![[[\alpha]]\varphi, \rho]\!] = \{p \mid \forall q.\ p \stackrel{\alpha}{\Rightarrow} q \text{ implies } q \in [\![\varphi, \rho]\!]\}.$$

The main aim of this section is to extend the maximally-expressive monitorable subset of μHML that was identified in [21, 22] using the framework developed in Section 3.

5.1 External Monitoring with Conditions

We define the external monitoring system with conditions similarly to Section 3. The syntax of Definition 8 is extended so that, for any instance of C, if m is a monitor and c a condition from C, then $c.m$ is a monitor.

Definition 11. *An* external C-monitor *on* Act *is defined by the grammar:*

$$m, n \in \mathrm{Mon}_e^C ::= \mathit{end} \mid \mathit{no} \mid \alpha.m \mid c.m \mid m + n \mid \mathit{rec}\ x.m \mid x,$$

where $c \in C$. *C-monitor behaviour is defined as in Table 1, but extending rule* mAct *to condition prefixes that generate condition actions (i.e.,* μ *ranges over* $\mathrm{Act} \cup C$*). We call the resulting monitor LTS* M_e^C.

For the instrumentation relation called I_e^C*, we consider the rules* iMon, iTer *from Table 1 for external actions* α *instead of the general action* μ*, rule* iAbs *from the same table, and rule* iCon *from Section 3.* ■

Note that the monitoring system (M_e^C, I_e^C) may be used to detect τ-transitions *implicitly*—we conjecture that this cannot be avoided in general. Consider two conflicting conditions c_1 and c_2, *i.e.,* $[\![c_1]\!] \cap [\![c_2]\!] = \emptyset$. Definition 11 permits monitors of the form $c_1.c_2.m$ that encode the fact that state m can only be reached when the system under scrutiny performs a non-empty sequence of τ-moves to transition from a state satisfying c_1 to another state satisfying c_2. This, in some sense, is also related to obscure silent action monitoring studied in [2].

We identify the grammar for the maximally-expressive monitorable syntactic subset of the logic WμHML. It uses the formula $[[\varepsilon]]\varphi$ defined as:

$$[[\varepsilon]]\varphi \equiv \mathsf{max}\ X.(\varphi \wedge [\tau]X)$$

The modality $[[\varepsilon]]\varphi$ quantifies universally over the set of processes that can be reached from a given one via any number of silent steps. Together with its dual $\langle\langle\varepsilon\rangle\rangle\varphi$ modality, $[[\varepsilon]]\varphi$ is used in the modal characterisation of weak bisimilarity [30, 34], in which τ transitions from one process may be matched by a (possibly empty) sequence of τ transitions from another.

Definition 12. *The weak safety fragment of* WμHML *with C is defined as:*

$$\varphi, \psi \in \text{WsHML}^C ::= \mathit{tt} \mid \mathit{ff} \mid [[\alpha]]\varphi \mid [[\varepsilon]](\neg c \vee \varphi) \mid \varphi \wedge \psi \mid \mathsf{max}\ X.\varphi \mid X,$$

where $c \in C$. ■

Theorem 4. *The monitoring system (M_e^C, I_e^C) monitors for* WsHML^C. □

We highlight the need to insulate the appearance of the implication $\neg c \vee \varphi$ from internal system behaviour by using the modality $[[\varepsilon]]$ in Definition 12. For conditions that are invariant under τ-transitions, this modality is not required but it cannot be eliminated otherwise; we revisit this point in Example 2.

5.2 Instantiating External Monitors with Conditions

We consider three different instantiations to our parametric external monitoring system of Section 5.1.

Recursion-free formulas The weak version of HML, denoted by wHML, is the recursion-free fragment of WμHML. Similarly to what was argued earlier in Section 4.2, it is an appropriate set of conditions to instantiate set C in WsHML^C, and the maximal monitorable fragment of WμHML with conditions from wHML is WsHML$^{w\text{HML}}$, defined by the following grammar, where $\eta \in w\text{HML}$:

$$\varphi, \psi ::= \mathsf{tt} \mid \mathsf{ff} \mid [[\alpha]]\varphi \mid [[\varepsilon]](\eta \vee \varphi) \mid \varphi \wedge \psi \mid \mathsf{max}\ X.\varphi \mid X.$$

Proposition 3. *The monitoring system $(M_e^{w\text{HML}}, I_e^{w\text{HML}})$ monitors for the logical fragment* WsHML$^{w\text{HML}}$. □

An important observation (that is perhaps surprising) is that WsHML$^{w\text{HML}}$ is *not* a fragment of WμHML, as the following example demonstrates.

Example 2. Although for any (closed) WsHML formula φ we have the logical equivalence $[[\varepsilon]]\varphi \equiv \varphi$ (notice that the monitor for φ that is guaranteed by Theorem 2 also monitors for $[[\varepsilon]]\varphi$), this logical equivalence does not hold for a formula φ from WμHML. Consider the formula φ_ϵ below that may be expressed using a formula from WsHML$^{w\text{HML}}$:

$$\varphi_\epsilon = [[\varepsilon]]\langle\langle\alpha\rangle\rangle\mathsf{tt} \equiv [[\varepsilon]](\langle\langle\alpha\rangle\rangle\mathsf{tt} \vee \mathsf{ff}) \in \text{WsHML}^{w\text{HML}}.$$

Formula φ_ϵ is not equivalent to $\langle\langle\alpha\rangle\rangle\mathsf{tt}$ (*e.g.* the process $\alpha.\mathsf{nil} + \tau.\mathsf{nil}$ satisfies $\langle\langle\alpha\rangle\rangle\mathsf{tt}$, but not φ_ϵ) meaning that $[[\varepsilon]]$ plays a discerning role in the context of WμHML. Furthermore, φ_ϵ holds for process $\tau.\alpha.\mathsf{nil}$, but not for $\alpha.\mathsf{nil}+\tau.\mathsf{nil}$, even though these two processes cannot be distinguished by *any* WμHML formula. In fact, it turns out that they are bisimilar with respect to *weak external transitions* and this bisimulation characterises the satisfaction of WμHML formulas [24]. Thus, there is no formula in WμHML that is equivalent to φ_ϵ. ■

Previous Runs and Alternating Monitoring A monitoring system could reuse information from previous system runs, perhaps recorded as execution logs, and whenever (sub)traces can be associated with specific states of the system, these can also be used as an instantiation for our parametric framework. More concretely, in [21, 22] it is shown that traces can be used to characterise the violation of WsHML formulas, or the satisfaction of formulas from the dual fragment, WcHML, defined below.

Definition 13. *The* co-safety WμHML *is defined by the grammar:*

$$\pi, \kappa \in \text{WcHML} ::= \mathit{tt} \mid \mathit{ff} \mid \langle\langle\alpha\rangle\rangle\theta \mid \theta \vee \chi \mid \mathit{min}\ X.\theta \mid X \quad ■$$

The witnessed rejection and acceptance traces can in turn be used as part of an augmented trace for an instantiation for C to obtain the monitorable dual logics WsHML$^{\text{WcHML}}$ and WcHML$^{\text{WsHML}}$ that alternate between rejection monitoring and acceptance monitoring. The logic WsHML$^{\text{WcHML}}$ is defined by the following grammar, where $\theta \in$ WsHML:

$$\varphi, \psi ::= \mathsf{tt} \mid \mathsf{ff} \mid [[\alpha]]\varphi \mid [[\varepsilon]](\theta \vee \varphi) \mid \varphi \wedge \psi \mid \mathsf{max}\ X.\varphi \mid X;$$

and WcHML$^{\text{WsHML}}$ is defined by the following grammar, where $\chi \in$ WcHML:

$$\pi, \kappa ::= \mathsf{tt} \mid \mathsf{ff} \mid \langle\langle\alpha\rangle\rangle\pi \mid \langle\langle\varepsilon\rangle\rangle(\chi \wedge \pi) \mid \pi \vee \kappa \mid \mathsf{min}\ X.\varphi \mid X.$$

Proposition 4. *The monitoring system* $(M_e^{\text{WcHML}}, I_e^{\text{WcHML}})$ *rejection-monitors for the logical fragment* WsHML$^{\text{WcHML}}$. □

One should observe that in this case, WsHML$^{\text{WcHML}}$ *is* a fragment of WμHML, in contrast to the previous instantiation WsHML$^{w\text{HML}}$ from Section 5.2.

Lemma 1. *For every* $[[\varepsilon]](\eta \vee \varphi) \in \text{WsHML}^{\text{WcHML}}$ *(where* $\eta \in \text{WsHML}$*), we have* $[[\varepsilon]](\eta \vee \varphi) \equiv \eta \vee \varphi$. □

Corollary 1. *For every formula in* $\text{WsHML}^{\text{WcHML}}$*, there is a logically equivalent formula in* WμHML. □

This entails that $\text{WsHML}^{\text{WcHML}}$ can be reformulated using the following, simpler, grammar (here $\eta \in \text{WsHML}$) which is clearly a fragment of WμHML:

$$\varphi, \psi ::= \mathsf{tt} \mid \mathsf{ff} \mid [[\alpha]]\varphi \mid \eta \vee \varphi \mid \varphi \wedge \psi \mid \mathsf{max}\ X.\varphi \mid X.$$

If the monitoring system can use such information from previous runs, there is no reason to limit this information to just one previous run. If the instrumentation mechanism can record up to i prior runs, the monitorable logic may be described as WsHML^{i+1}, defined inductively in the following way:

- $\text{WsHML}^1 = \text{WsHML}$ and $\text{WcHML}^1 = \text{WcHML}$; and
- $\text{WsHML}^{i+1} = \text{WsHML}^{\text{WcHML}^i}$ and $\text{WcHML}^{i+1} = \text{WcHML}^{\text{WsHML}^i}$.

Whenever this setup can be extended to unlimited prior runs, the resulting rejection-monitorable fragment would be $\text{WsHML}^\omega = \bigcup_i \text{WsHML}^i$, which is also described by the following grammar:

$$\varphi, \psi ::= \mathsf{tt} \mid \mathsf{ff} \mid [[\alpha]]\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \mathsf{max}\ X.\varphi \mid X.$$

WsHML^ω is a non-trivial extension of WsHML *which is still within* WμHML.

Failure to Execute an Action and Refusals In Subsection 4.1, we instantiated the condition set C as the set of formulas from μHML that assert the inability of a process to perform an action. These formulas are of the form $[\alpha]\mathsf{ff}$. We recast this approach in the setting of weak monitorability. In this setting where the monitoring system and the specification formulas ignore any silent transitions, the inability of a process to perform an α-transition acquires a different meaning from the one used for the basic system. In particular, we consider a stronger version of these conditions that incorporates stability; this makes them invariant over τ-transitions. We say that p *refuses* α when $p \stackrel{\tau}{\not\rightarrow}$ and $p \stackrel{\alpha}{\not\rightarrow}$. In [31], a very similar notion is used for refusal testing (see also [1]). Thus, much in line with [31], we use the following definition.

Definition 14. *A process* p *of an LTS* L refuses *action* $\alpha \in \text{Act}$ *and write* $p\ \mathtt{ref}\ \alpha$ *when* $p \stackrel{\tau}{\not\rightarrow}_L$ *and* $p \stackrel{\alpha}{\not\rightarrow}_L$*. The set of conditions that corresponds to refusals is thus* $R_{\text{Act}} = \{[\tau]\mathit{ff} \wedge [\alpha]\mathit{ff} \mid \alpha \in \text{Act}\}$. ■

According to Theorem 4, the largest fragment of μHML that we can monitor for, using monitors that can detect refusals, is $\text{WsHML}^{R_{\text{Act}}}$, given by the following grammar:

$$\varphi, \psi ::= \mathsf{tt} \mid \mathsf{ff} \mid [[\alpha]]\varphi \mid [[\varepsilon]](\langle\tau\rangle\mathsf{tt} \vee \langle\alpha\rangle\mathsf{tt} \vee \varphi) \mid \varphi \wedge \psi \mid \mathsf{max}\ X.\varphi \mid X.$$

Again, $\langle\tau\rangle\mathsf{tt} \vee \langle\alpha\rangle\mathsf{tt} \vee \varphi$ is best read as the implication $([\tau]\mathsf{ff} \wedge [\alpha]\mathsf{ff}) \rightarrow \varphi$: if the process is stable and cannot perform an α-transition, then φ must hold.

Proposition 5. *The monitoring system* $(M_e^{R_{\text{Act}}}, I_e^{R_{\text{Act}}})$ *monitors for the logical fragment* $\text{WsHML}^{R_{\text{Act}}}$. □

Example 3. Consider the formula

$$\varphi_s = [[\varepsilon]](\langle\tau\rangle\mathsf{tt} \vee \langle\alpha\rangle\mathsf{tt} \vee [[\beta]]\mathsf{ff}) \in \text{WsHML}^{R_{\text{Act}}}.$$

Formula φ_s claims that at every stable state that the system can reach, if action α is impossible, then action β should also be impossible. We can see that φ_s is true for $\tau.\mathsf{nil} + \beta.\mathsf{nil}$, but not for $\beta.\mathsf{nil}$. However, the two processes cannot be distinguished by WμHML, as they have the same weak external transitions. Therefore, $\text{WsHML}^{R_{\text{Act}}}$ is not a fragment of WμHML — but, as we have seen, it is a fragment of μHML. Here we have a part of the formula that clearly is not part of WμHML. That is $\langle\tau\rangle\mathsf{tt}$, which asserts that the process can perform a silent transition. ■

Example 4. Let us consider an LTS L_0 of stable processes — that is, L_0 is an LTS without any silent transitions. L_0 offers a simplified setting to cast our observations. In this case, the $[[\varepsilon]]$, $[\tau]$, and $\langle\tau\rangle$ modalities can be eliminated from our formulas, and weak modalities are equivalent to strong modalities. This allows us to simplify the grammar for $\text{WsHML}^{F_{\text{Act}}}$ as follows:

$$\begin{array}{llllll} \varphi, \psi ::= & \mathsf{tt} & \mid \mathsf{ff} & \mid [\alpha]\varphi & \mid \langle\alpha\rangle\mathsf{tt} \vee \varphi \\ \mid & \varphi \wedge \psi & \mid \mathsf{max}\ X.\varphi & \mid X. & \end{array}$$

Perhaps unsurprisingly, this grammar yields the same formulas as the restriction of grammar of Subsection 4.1 on external actions. An instance of a specification that can be formalized in this fragment is the following. Consider a simple server-client system, where the client can request a resource, which is represented by action `rq`, and the server may give a positive response, represented by action `rs`, after which it needs to allocate said resource to the client, represented by action `al`. A reasonable specification for the server is that if it is impossible at the moment to provide a resource, then it should not give a positive response to the client. In the above simplification of $\text{WsHML}^{F_{\text{Act}}}$, this specification can be formalized as $[\mathtt{rq}](\langle\mathtt{al}\rangle\mathsf{tt} \vee [\mathtt{rs}]\mathsf{ff})$. If the LTS includes silent transitions, the corresponding specification would be written as

$$\varphi_r = [\mathtt{rq}][[\varepsilon]](\langle\tau\rangle\mathsf{tt} \vee \langle\mathtt{al}\rangle\mathsf{tt} \vee [[\mathtt{rs}]]\mathsf{ff}).$$

In other words, after a request, if the server cannot provide a resource and it is stable — so, there is no possibility that after some time the resource will be available — then the server should not give a positive response to the client. ■

6 Conclusions

In order to devise effective verification strategies that straddle between the pre- and post-deployment phases of software production, one needs to understand

better the monitorability aspects of the correctness properties that are to be verified. We have presented a general framework that allows us to determine maximal monitorable fragments of an expressive logic that is agnostic of the verification technique employed, namely μHML. By way of a number of instantiations, we also show how the framework can be used to reason about the monitorability induced by various forms of augmented traces. Our next immediate concern is to validate the proposed instantiations empirically by constructing monitoring systems and tools that are based on these results, as we did already for the original monitorability results of [21, 22] in [9, 10, 12].

Related Work Monitorability for μHML was first examined in [21, 22]. This work introduced the external monitoring system and identified WsHML as the largest monitorable fragment of μHML, with respect to that system. The ensuring work in [2] focused on monitoring setups that can distinguish silent actions to a varying degree, and introduced the basic monitoring system, showing analogous monitorability results for μHML.

Monitorability has also been examined for languages defined over traces, such as LTL. Pnueli and Zaks in [32] define a notion of monitorability over traces, although they do not attempt maximal monitorability results. Diekert and Leuckert revisited monitorability from a topological perspective in [16]. Falcone *et al.* in [17] extended the work in [32] to incorporate enforcement and introduced a notion of monitorability on traces that is parameterized with respect to a truth domain that corresponds to our separation to acceptance- and rejection-monitorable properties. In [13], the authors use a monitoring system that can generate derivations of satisfied formulas from a fragment of LTL. However, they do not argue that this fragment is somehow maximal. There is a significant body of work on synthesizing monitors from LTL formulas, *e.g.* [13, 23, 33, 35], and it would be worth investigating whether our general techniques for monitor synthesis can be applied effectively in these cases.

Phillips introduced *refusal testing* in [31] as a way to extend the capabilities of testing (see [18] for a discussion on how our monitoring setup relates to testing preorders). The meaning of refusals in [31] is very close to the one in Definition 14 and it is interesting to note how Phillips' use of tests for refusal formulas is similar to our monitoring mechanisms for refusals. Abramsky [1] uses refusals in the context of a much more powerful testing machinery, in order to identify the kind of testing power that is required for distinguishing non-bisimilar processes.

The decomposition of the verification burden across verification techniques, or across iterations of alternating monitoring runs as presented in Section 5, can be seen as a method for *quotienting*. In [7] Andersen studies quotienting of the specification logics discussed in this paper to reduce the state-space during model checking and thus increase its efficiency (see also [27] for a more recent treatment). The techniques used rely heavily on the model's concurrency constructs and may produce formulas that are larger in size than the original, but which can be checked against a smaller component of the model. In multi-pronged approaches to verification one would expect to encounter similar difficulties occasionally.

References

1. Abramsky, S.: Observation equivalence as a testing equivalence. Theoretical Computer Science 53(2-3), 225–241 (1987)
2. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A.: Monitoring for silent actions. In: 37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017. (to appear)
3. Aceto, L., Achilleos, A., Francalanza, A., Ingólfsdóttir, A., Kjartansson, S.Ö.: Determinizing monitors for HML with recursion. CoRR abs/1611.10212 (2016)
4. Aceto, L., Ingólfsdóttir, A., Larsen, K.G., Srba, J.: Reactive Systems: Modelling, Specification and Verification. Cambridge University Press, New York, NY, USA (2007)
5. Ahrendt, W., Chimento, J.M., Pace, G.J., Schneider, G.: A specification language for static and runtime verification of data and control properties. In: Bjørner, N., deBoer, F. (eds.) FM 2015: Formal Methods. LNCS, vol. 9109, pp. 108–125. Springer (2015)
6. Aktug, I., Naliuka, K.: ConSpec - A formal language for policy specification. Science of Computer Programming 74(1-2), 2–12 (2008)
7. Andersen, H.R.: Partial model checking (extended. In: Proceedings of Tenth Annual IEEE Symposium on Logic in Computer Science. pp. 398–407. IEEE (1995)
8. Artho, C., Barringer, H., Goldberg, A., Havelund, K., Khurshid, S., Lowry, M.R., Pasareanu, C.S., Rosu, G., Sen, K., Visser, W., Washington, R.: Combining test case generation and runtime verification. Theoretical Computer Science 336(2-3), 209–234 (2005)
9. Attard, D.P., Francalanza, A.: A Monitoring Tool for a Branching-Time Logic. In: Runtime Verification. LNCS, vol. 10012, pp. 473–481. Springer (2016)
10. Attard, D.P., Francalanza, A.: Trace partitioning and local monitoring for asynchronous components. In: Cimatti, A., Sirjani, M. (eds.) Software Engineering and Formal Methods - 15th International Conference, SEFM 2017, Trento, Italy, September 4-8, 2017, Proceedings. LNCS, vol. 10469, pp. 219–235. Springer (2017)
11. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 193–207. TACAS '99, Springer-Verlag, London, UK (1999)
12. Cassar, I., Francalanza, A.: On implementing a monitor-oriented programming framework for actor systems. In: integrated Formal Methods (iFM). pp. 176–192. Springer (2016)
13. Cini, C., Francalanza, A.: An LTL Proof System for Runtime Verification. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS). vol. 9035, pp. 581–595. Springer (2015)
14. Decker, N., Leucker, M., Thoma, D.: jUnitRV–Adding runtime verification to jUnit. In: NASA FM. Lecture Notes in Computer Science, vol. 7871, pp. 459–464. Springer Berlin Heidelberg (2013)
15. Desai, A., Dreossi, T., Seshia, S.A.: Combining model checking and runtime verification for safe robotics. In: Lahiri, S., Reger, G. (eds.) Runtime Verification: 17th International Conference, RV 2017, Seattle, WA, USA, September 2017. pp. 172–189. LNCS, Springer (2017)
16. Diekert, V., Leucker, M.: Topology, monitorable properties and runtime verification. Theoretical Computer Science 537, 29–41 (2014)
17. Falcone, Y., Fernandez, J.C., Mounier, L.: What can you verify and enforce at runtime? Int. J. Software Tools Technol. Trans. 14(3), 349–382 (2012)

18. Francalanza, A.: A Theory of Monitors. In: Jacobs, B., Löding, C. (eds.) Foundations of Software Science and Computation Structures: 19th International Conference (FoSSaCS). LNCS, vol. 9634, pp. 145–161. Springer (2016)
19. Francalanza, A.: Consistently-detecting monitors. In: Meyer, R., Nestmann, U. (eds.) 28^{th} International Conference on Concurrency Theory (CONCUR 2017). LIPIcs, vol. 85, pp. 8:1–8:19. Schloss Dagstuhl, Dagstuhl, Germany (2017)
20. Francalanza, A., Aceto, L., Achilleos, A., Attard, D.P., Cassar, I., Monica, D.D., Ingólfsdóttir, A.: A Foundation for Runtime Monitoring. In: Lahiri, S.K., Reger, G. (eds.) Runtime Verification: 17th International Conference, RV 2017, Seattle, WA, USA, September 2017. LNCS, vol. 10548, pp. 8–29. Springer (2017)
21. Francalanza, A., Aceto, L., Ingólfsdóttir, A.: On verifying Hennessy-Milner logic with recursion at runtime. In: Bartocci, E., Majumdar, R. (eds.) Runtime Verification - 15th International Conference, RV 2015 Vienna, Austria, September 2015. LNCS, vol. 9333, pp. 71–86. Springer (2015)
22. Francalanza, A., Aceto, L., Ingolfsdottir, A.: Monitorability for the Hennessy–Milner Logic with recursion. Formal Methods in System Design (FMSD) 51(1), 87–116 (2017)
23. Geilen, M.: On the construction of monitors for temporal logic properties. Electron. Notes Theor. Comput. Sci. 55(2), 181–199 (2001)
24. Hennessy, M., Milner, R.: Algebraic Laws for Nondeterminism and Concurrency. Journal of the ACM 32(1), 137–161 (1985)
25. Kejstová, K., Ročkai, P., Barnat, J.: From Model Checking to Runtime Verification and Back. In: Runtime Verification: 17th International Conference, RV 2017, Seattle, WA, USA, September 2017. pp. 225–240. Springer (2017)
26. Kozen, D.: Results on the propositional μ-calculus. Theoretical Computer Science 27(3), 333–354 (1983)
27. Lang, F., Mateescu, R.: Partial model checking using networks of labelled transition systems and boolean equation systems. Logical Methods in Computer Science 9(4) (2013)
28. Larsen, K.G.: Proof systems for satisfiability in Hennessy-Milner logic with recursion. Theoretical Computer Science 72(2), 265–288 (1990)
29. Leucker, M., Schallhart, C.: A brief account of Runtime Verification. The Journal of Logic and Algebraic Programming 78(5), 293–303 (2009)
30. Milner, R.: Communication and Concurrency. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1989)
31. Phillips, I.: Refusal testing. Theoretical Computer Science 50(3), 241–284 (1987)
32. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: FM 2006: Formal Methods. pp. 573–586. Springer (2006)
33. Sen, K., Roşu, G., Agha, G.: Generating Optimal Linear Temporal Logic Monitors by Coinduction. In: Saraswat, V.A. (ed.) Advances in Computing Science (ASIAN). Progamming Languages and Distributed Computation. pp. 260–275. Springer (2003)
34. Stirling, C.: Modal and Temporal Properties of Processes. Springer-Verlag New York, Inc., New York, NY, USA (2001)
35. Vardi, M.Y.: An Automata-Theoretic approach to Linear Temporal Logic. In: Logics for Concurrency: Structure versus Automata. LNCS, vol. 1043, pp. 238–266. Springer-Verlag (1996)