If At First You Don't Succeed: Extended Monitorability through Multiple Executions

Antonis Achilleos
Reykjavik University
Reykjavik, Iceland
antonios@ru.is

Adrian Francalanza
University of Malta
Msida, Malta
adrian.francalanza@um.edu.mt

Jasmine Xuereb
Reykjavik University and University of Malta
Reykjavik, Iceland, and Msida, Malta
jasmine.xuereb.15@um.edu.mt

***Abstract*—This paper studies the extent to which branching-time properties can be adequately verified using runtime monitors. We depart from the classical setup where monitoring is limited to a single system execution and investigate the enhanced observational capabilities when monitoring a system over multiple runs. To ensure generality, we focus on branching-time properties expressed in the modal μ-calculus, a well-studied foundational logic. Our results show that the proposed setup can systematically extend established monitorability limits for branching-time properties. We validate our results by instantiating them to verify actor-based systems. We also prove bounds that capture the correspondence between the syntactic structure of a property and the number of required system runs.**

***Index Terms*—Runtime verification, Branching-time logics, Monitorability**

I. Introduction

Branching-time properties have long been considered the preserve of static analyses, verified using established techniques such as model checking [1], [2]. Unfortunately, these verification techniques cannot be used when the system model is either too expensive to build and analyse (*e.g.* state-explosion problems), poorly understood (*e.g.* system logic governed by machine-learning procedures) or downright unavailable (*e.g.* restrictions due to intellectual property rights). Recent work has shown that runtime monitoring can be used effectively (in isolation or in conjunction with other verification techniques) to verify certain branching-time properties [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16]. Specifically, (execution) *monitors* (or sequence recognisers) [17], [18], [19], [20] passively observe the *execution* of a system-under-scrutiny (SUS), possibly aided by auxiliary information, to compare the observed behaviour (instead of its state space) against a correctness property of interest.

The use of monitors for verification purposes is called runtime verification (RV) [21], [22]. It is weaker than static techniques for verifying both linear-time and branching-time properties: monitor observations are constrained to the current (single) computation path of the SUS limiting the *range* of verifiable properties. For instance, the linear-time property $G\,\psi$ (*i.e.,* ψ must always hold) can only be monitored for violations but not satisfactions, whereas infinite renewal properties such as $GF\,\psi$ (*i.e.,* always, eventually ψ) *cannot* be monitored for at all. Monitorability limits are more acute for branching-time properties: the *maximal* monitorable subset for the modal μ-calculus was shown to be semantically equivalent to the syntactic fragment sHML$\cup$cHML [8], [11].

Example I.1. Consider a server SUS exhibiting four events: receive queries (r), service queries (s), allocate memory (a) and close connection (c). Modal μ-calculus properties[1] such as *"all interactions can only start with a receive query"*, *i.e.,* $\varphi_0 \stackrel{\text{def}}{=} [s]\mathsf{ff} \wedge [a]\mathsf{ff} \wedge [c]\mathsf{ff} \in$ sHML can be runtime verified since any SUS execution observed that starts with event s, a or c confirms that the running SUS violates the property (irrespective of any execution events that may follow). However, the branching-time property *"systems that can perform a receive action,* $\langle r\rangle\mathsf{tt}$, cannot *also close,* $[c]\mathsf{ff}$*"*, *i.e.,*

$$\varphi_1 \stackrel{\text{def}}{=} \langle r\rangle\mathsf{tt} \Rightarrow [c]\mathsf{ff} \quad\equiv\quad [r]\mathsf{ff} \vee [c]\mathsf{ff} \quad\notin\quad \text{sHML} \cup \text{cHML}$$

is *not* monitorable with respect to either satisfactions or violations. No (single) trace prefix provides enough evidence to conclude that a system satisfies this property, whereas an observed trace starting with r (dually c) is not enough to conclude that the emitting SUS (state) violates the property: one would also need evidence that the same state can also emit c (dually r). ■

There are various approaches for extending the set of monitorable properties. One method is to weaken the detection requirements expected of the monitors [23], [24] effecting the verification (*e.g.* by allowing certain violations to go undetected). This, in turn, impinges on what it means for a property to be monitorable. Another approach is to increase the monitors' observational capabilities. Aceto *et al.* [25] investigate the increase in monitor observational power after augmenting the information recorded in the trace: apart from reporting computational steps *that happened*, they consider trace events that can also record branching information such as the computation steps that *could have* happened *at a particular state*, or the computation steps that could *not* have happened.

Supported by the doctoral student grant of the Reykjavik University Reseach Fund, by the grant "Mode(l)s of Verification and Monitorability" (MoVeMent) (grant no 217987) of the Icelandic Research Fund, and by the grants "Security Behavioural APIs" (grant no I22LU01) and "Runtime Monitoring for IoT" (grant no CPSRP01-25) of University of Malta Research Seed Fund.

[1] Formula $[\alpha]\mathsf{ff}$ describes states that *cannot* perform α transitions whereas its dual, $\langle\alpha\rangle\mathsf{tt}$ describes states that *can* perform α transitions.

This approach treats the SUS as a *grey-box* [13], [26] since the augmented traces reveal *specific* system information about the SUS states reached. This paper builds on Aceto *et al.*'s work while sticking, as much as possible, to a black-box treatment of the SUS. We study the increase in observational power obtained from considering multiple execution traces for the same SUS *without* relying directly on information about the specific intermediary states reached during monitoring.

Example I.2. Property φ_1 from Ex. I.1 can be monitored for violations over *two* executions of the same system: a first trace starting with event, r, and a second trace starting with event, c, is sufficient evidence to conclude that the SUS violates φ_1. ■

Analysing multiple traces is *not* always sufficient to conclude that a system violates a property with disjunctions since the same prefix could, in principle, reach different states.

Example I.3. Consider the property *"after any receive query,* $[r]\ldots$*, if a SUS can service it,* $\langle s\rangle\mathtt{tt}$*, then (it takes precedence and) it should not allocate more memory,* $[a]\mathtt{ff}$*"*, expressed as

$$\varphi_2 \stackrel{\text{def}}{=} [r](\langle s\rangle\mathtt{tt} \Rightarrow [a]\mathtt{ff}) \;\equiv\; [r]([s]\mathtt{ff} \vee [a]\mathtt{ff})$$

Intuitively, φ_2 is violated when the state reached after event r can perform *both* events s and a. Observing traces $rs\cdots$ and $ra\cdots$ along two executions is not enough to conclude that the SUS violates φ_2: although both executions start from the same state, say p, distinct intermediary states could be reached after event r, *i.e.,* $p \xrightarrow{r} p_1 \xrightarrow{s} p_2$ and $p \xrightarrow{r} p_1' \xrightarrow{a} p_2'$ where $p_1 \neq p_1'$. ■

Although non-deterministic SUS behaviour cannot be ruled out in general, many systems are deterministic w.r.t. a *subset* of actions, such as asynchronous LTSs and output actions [27], [28] (*e.g.* if r was an asynchronous output in Ex. I.3 then $p_1 = p_1'$.) Moreover, deterministic behaviour is *not* necessarily required to runtime-verify all the behaviours specified.

Example I.4. Consider the property that, in addition to the behaviour described by φ_2, it requires that *"...the SUS does* not *exhibit any action after a close event"*, formalised as φ_3.

$$\varphi_3 \stackrel{\text{def}}{=} \big([r]([s]\mathtt{ff} \vee [a]\mathtt{ff})\big) \;\wedge\; \big([c]([r]\mathtt{ff} \wedge [s]\mathtt{ff} \wedge [a]\mathtt{ff} \wedge [c]\mathtt{ff})\big)$$

It might be reasonable to assume that a SUS behaves deterministically for receive actions (*e.g.* when a single thread is in charge of receiving). Moreover, *no* determinism assumption is required for close actions to runtime verify the subformula $[c]([r]\mathtt{ff} \wedge [s]\mathtt{ff} \wedge [a]\mathtt{ff} \wedge [c]\mathtt{ff})$; any trace from either $cr\cdots, cs\cdots$, $ca\cdots$ or $cc\cdots$ suffices to infer the violation of φ_3. ■

The properties discussed in this paper are formalised in terms of a variant of the modal μ-calculus [29] called Hennessy-Milner Logic with Recursion [30], RECHML. This logic is a natural choice for describing branching-time properties and is employed by state-of-the-art model checkers, including mCRL2 [31] and UPPAAL [32], as well as detectEr [12], [33], a stable RV tool. It has been shown to embed standard logics such as LTL, CTL and CTL* [2], [1], [23]. Moreover, existing maximality results for branching-time logics [8], [25], [24] have only been established for RECHML. Our exposition focusses on "safety" properties that can be monitored for violations; monitoring for satisfactions of branching-time properties is symmetric [8]. This paper presents an augmented monitoring setup that repeatedly analyses a (potentially non-deterministic) SUS across multiple executions, so as to study how the monitorability limits established in [11], [8] are affected. Our contributions are:

1) A formalisation of a monitoring setup that gathers information over multiple system runs (Sec. III).
2) An analysis, formalised as a proof system, that uses sets of partial traces to runtime verify the system against a branching-time property (Sec. III).
3) A definition formalising what it means for a monitor to correctly analyse a property over multiple runs (Sec. IV) and, dually, what it means for a property to be monitorable over multiple runs (Sec. V).
4) The identification of an *extended* logical fragment that is monitorable over the augmented monitoring setup handling multiple runs (Sec. V), and the establishment that the extended fragment is maximally expressive (Sec. V).
5) An instantiation of the multi-run RV framework to actor-based systems (Sec. VI), a popular concurrency paradigm.
6) A method for systematically determining the number of SUS executions required to conduct RV from the syntactic structure of the formula being verified (Sec. VII).

Full proofs and additional explanations and examples may be found in the companion technical report [34].

II. PRELIMINARIES

We assume a set of actions, $\eta, \xi \in \text{ACT} = \text{TACT} \uplus \{\tau\}$, with a distinguished *silent (untraceable) action* τ and a set of *traceable actions*, $\mu, \lambda \in \text{TACT} = \text{EACT} \uplus \text{IACT}$, that consists of two disjoint sets. *External actions*, $\alpha, \beta \in \text{EACT}$, describe computation steps observable to an outside entity which are the subject of correctness specifications. *Internal actions*, $\gamma, \delta \in \text{IACT}$, are not of concern to correctness specifications but can still be discerned by a monitor with the appropriate instrumentation mechanism. Notably, silent actions (denoted by τ) cannot be traced by monitors.

A SUS is modelled as an *Instrumentable Labelled Transition System* (ILTS), a septuple of the form

$$\langle \text{PRC}, \equiv, \text{EACT}, \text{IACT}, \{\tau\}, \rightarrow, \text{DET}\rangle$$

SUS states are denoted by processes, $p, q \in \text{PRC}$, with an associated equivalence relation, $\equiv \subseteq \text{PRC} \times \text{PRC}$. The transition relation, $\longrightarrow \subseteq (\text{PRC} \times \text{ACT} \times \text{PRC})$, is defined over arbitrary actions (*i.e.,* silent, internal and external). We write $p \xrightarrow{\eta} q$ instead of $(p, \eta, q) \in \longrightarrow$, and $p \stackrel{\eta}{\not\longrightarrow}$ whenever $\nexists q$ such that $p \xrightarrow{\eta} q$. ILTS transitions abstract over equivalent states:

$$\text{for any } p \equiv q, \text{ if } p \xrightarrow{\eta} p' \text{ then there exists } q' \text{ such that } q \xrightarrow{\eta} q' \text{ where } p' \equiv q'.$$

Instrumentation also can abstract over (*non*-traceable) silent transitions because they are *confluent* w.r.t. other actions:

for any p, whenever $p \xrightarrow{\tau} p'$ and $p \xrightarrow{\eta} p''$ then,
either $\eta = \tau$ and $p' \equiv q'$, or there exists a state q and
transitions $p' \xrightarrow{\eta} q$ and $p'' \xrightarrow{\tau} q$ joining the diamond.

An ILTS partitions traceable actions via the predicate $\textsc{Det} : \textsc{tAct} \to \textsc{Bool}$ where all actions μ satisfying the predicate, $\textsc{Det}(\mu) = \mathsf{true}$, must be *deterministic* (up to $\equiv$):

$$\text{if } p \xrightarrow{\mu} p' \text{ and } p \xrightarrow{\mu} p'' \text{ then } p' \equiv p''.$$

Weak transitions, $p \Longrightarrow q$, abstract over both silent and internal actions whereas *weak traceable transition*, $p \Longrightarrow_{\tau} q$, abstract over silent actions only. Thus, $p \Longrightarrow q$ holds when $p = q$ or $\exists p'$ and $\eta \in \big(\{\tau\} \cup \textsc{iAct}\big)$ such that $p \xrightarrow{\eta} p' \Longrightarrow q$. Analogously, $p \Longrightarrow_{\tau} q$ holds if $p = q$ or $\exists p'$ such that $p \xrightarrow{\tau} p' \Longrightarrow_{\tau} q$. We write $p \overset{\alpha}{\Longrightarrow} q$ when $\exists p', p''$ such that $p \Longrightarrow p' \xrightarrow{\alpha} p'' \Longrightarrow q$, and write $p \overset{\mu}{\Longrightarrow}_{\tau} q$ when $\exists p', p''$ such that $p \Longrightarrow_{\tau} p' \xrightarrow{\mu} p'' \Longrightarrow_{\tau} q$. Actions can be sequenced to form *traces*, $t, u \in \textsc{Trc} = \textsc{tAct}^*$, representing prefixes of system runs. A trace with action μ at its head and continuation t is denoted as μt, whereas a trace with prefix t and action μ at its end is denoted as $t\mu$. For $t = \mu_1 \cdots \mu_n$, we write $p \overset{t}{\Longrightarrow}_{\tau} q$ instead of the sequence of transitions $p \overset{\mu_1}{\Longrightarrow}_{\tau} \cdots \overset{\mu_n}{\Longrightarrow}_{\tau} q$. A system (state) p *produces* a trace t when $\exists q$ such that $p \overset{t}{\Longrightarrow}_{\tau} q$. The set of all the traces produced by the state p is denoted by T_p. *Histories* $H \in \textsc{Hst}$ where $\textsc{Hst} \subseteq \textsc{Trc}$ are finite sets of traces where H, t is shorthand for the disjoint union $H \uplus \{t\}$.

Remark 1. An ILTS provides two (global) views of a SUS: an external one, as viewed by an observer limited to EAct, and a lower-level view as seen by an instrumented monitor privy to tAct and Det. The SUS treatment is still considered *black-box* for two reasons. First, the information on traceable actions tAct (needed anyway for instrumentation reasons) together with their deterministic properties Det, characterises an infinite class of systems, not one specific SUS. Second, ILTS determinism guarantees permit a monitor to reason about states within the same equivalence class (*i.e.,* $p \equiv q$), not directly on specific states. ILTSs do not limit the range of systems modelled. Deterministic systems can be described by requiring $\textsc{Det}(\mu)$=true for all actions, whereas general systems would have $\textsc{Det}(\mu)$=false. Silent actions capture β-moves [35], [36] which arise naturally in linearly-typed system and as thread-local moves in concurrent systems. ■

Properties are formulated for the external SUS view in terms of recHML formulae. This logic is defined by the negation free grammar in Fig. 1, which assumes a countably infinite set of formula variables $\mathsf{X}, \mathsf{Y}, \ldots \in \textsc{TVars}$. Apart from the standard constructs for truth, falsity, conjunction and disjunction, the logic includes existential and universal modalities that operate over the *external* actions EAct. Least and greatest fixed points, $\mathsf{min}\mathsf{X}.\varphi$ and $\mathsf{max}\mathsf{X}.\varphi$ respectively, bind free instances of variable X in φ. We assume standard definitions

recHML Syntax

$\varphi, \psi \in \textsc{recHML} ::=$		X	(rec. variable)
$\mid \mathsf{tt}$	(truth)	$\mid \langle\alpha\rangle\varphi$	(existential modality)
$\mid \mathsf{ff}$	(falsehood)	$\mid [\alpha]\varphi$	(universal modality)
$\mid \varphi \wedge \psi$	(conjunction)	$\mid \mathsf{min}\mathsf{X}.\varphi$	(least fixed point)
$\mid \varphi \vee \psi$	(disjunction)	$\mid \mathsf{max}\mathsf{X}.\varphi$	(greatest fixed point)

Branching-Time Semantics

$$[\![\mathsf{tt}, \rho]\!] \overset{\text{def}}{=} \textsc{Prc} \qquad [\![\mathsf{ff}, \rho]\!] \overset{\text{def}}{=} \emptyset$$

$$[\![\varphi \vee \psi, \rho]\!] \overset{\text{def}}{=} [\![\varphi, \rho]\!] \cup [\![\psi, \rho]\!] \qquad [\![\varphi \wedge \psi, \rho]\!] \overset{\text{def}}{=} [\![\varphi, \rho]\!] \cap [\![\psi, \rho]\!]$$

$$[\![[\alpha]\varphi, \rho]\!] \overset{\text{def}}{=} \big\{ p \mid \forall q \cdot p \overset{\alpha}{\Longrightarrow} q \text{ implies } q \in [\![\varphi, \rho]\!] \big\}$$

$$[\![\langle\alpha\rangle\varphi, \rho]\!] \overset{\text{def}}{=} \big\{ p \mid \exists q \cdot p \overset{\alpha}{\Longrightarrow} q \text{ and } q \in [\![\varphi, \rho]\!] \big\}$$

$$[\![\mathsf{min}\mathsf{X}.\varphi, \rho]\!] \overset{\text{def}}{=} \bigcap \big\{ P \mid [\![\varphi, \rho[\mathsf{X} \mapsto P]]\!] \subseteq P \big\} \qquad [\![\mathsf{X}, \rho]\!] \overset{\text{def}}{=} \rho(\mathsf{X})$$

$$[\![\mathsf{max}\mathsf{X}.\varphi, \rho]\!] \overset{\text{def}}{=} \bigcup \big\{ P \mid P \subseteq [\![\varphi, \rho[\mathsf{X} \mapsto P]]\!] \big\}$$

Fig. 1. recHML in the Branching-Time Setting.

for open and closed formulae and work up to α-conversion, assuming formulae to be closed and guarded, unless otherwise stated. For formulae φ and ψ, and variable X, $\varphi[\psi/\mathsf{X}]$ denotes the substitution of all free occurrences of X in φ with ψ.

The denotational semantics function $[\![-]\!]$ in Fig. 1 gives a branching-time interpretation to recHML by mapping formulae to sets of system states, $[\![-]\!] : \textsc{recHML} \to \mathscr{P}(\textsc{Prc})$. This function is defined with respect to an *environment* ρ, which maps formula variables to sets of states, $\rho : \textsc{TVars} \to \mathscr{P}(\textsc{Prc})$. Given a set of states P, $\rho[\mathsf{X} \mapsto P]$ denotes the environment mapping X to P, mapping as ρ on all other variables. Existential modalities $\langle\alpha\rangle\varphi$ denote the set of system states that can perform *at least one* α-labelled (weak) transition and reach a state that satisfies the continuation φ. Conversely, universal modalities $[\alpha]\varphi$ denote the set of systems that reach states satisfying φ for *all* (possibly none) their α-transitions. The set of systems that satisfy least fixed point formulae (resp. greatest fixed point) is given by the intersection (resp. union) of all pre-fixed points (resp. post-fixed points) of the function induced by the corresponding binding formula. The remaining cases are standard. The interpretation of closed formulae is independent of ρ; we write $[\![\varphi]\!]$ in lieu of $[\![\varphi, \rho]\!]$. A state p *satisfies* φ if $p \in [\![\varphi]\!]$ and *violates* it if $p \notin [\![\varphi]\!]$; equivalent states satisfy (resp. violate) the same formulae, Prop. II.1. Two formulae φ and ψ are said to be *equivalent*, denoted as $\varphi \equiv \psi$, whenever $[\![\varphi]\!] = [\![\psi]\!]$. The negation of a formula can be obtained by duality in the usual way, *e.g.* $\neg(\mathsf{min}\mathsf{X}.\langle\alpha\rangle\mathsf{tt} \vee [\beta]\mathsf{X}) = \mathsf{max}\mathsf{X}.[\alpha]\mathsf{ff} \wedge \langle\beta\rangle\mathsf{X}$.

Proposition II.1 (Behavioural Equivalence)**.** *For all (closed) formulae* $\varphi \in \textsc{recHML}$*, if* $p \in [\![\varphi]\!]$ *and* $p \equiv q$ *then* $q \in [\![\varphi]\!]$. ■

Several logical formulae from Fig. 1 are not monitorable w.r.t. classical RV limited to one (partial) execution of the system. The safety subset of monitorable recHML formulae is characterised by the syntactic fragment sHML [37].

Theorem II.2 (Monitorability [8])**.** *Any* $\varphi \in$ RECHML *is monitorable (for violations) iff there exists* $\psi \in$ SHML *and* $\varphi \equiv \psi$*:*

$$\varphi, \psi \in \text{SHML} ::= \mathsf{tt} \mid \mathsf{ff} \mid [\alpha]\varphi \mid \varphi \wedge \psi \mid \max \mathsf{X}.\varphi \mid \mathsf{X} \quad \blacksquare$$

Example II.1. The property *"after any number of serviced queries,* $[r][s]\ldots$*, a state that can close a connection,* $\langle c\rangle\mathsf{tt}$*,* cannot *allocate memory,* $[a]\mathsf{ff}$*"* is *not* monitorable.

$$\begin{aligned}\varphi_4 \stackrel{\text{def}}{=}\ & \max \mathsf{X}.\big([r][s]\mathsf{X} \wedge (\langle c\rangle\mathsf{tt} \Rightarrow [a]\mathsf{ff})\big)\\ \equiv\ & \max \mathsf{X}.\big([r][s]\mathsf{X} \wedge ([c]\mathsf{ff} \vee [a]\mathsf{ff})\big)\end{aligned}$$

Specifically, a system violates φ_4 if it is capable of producing *both* actions a and c after an unbounded, but *finite*, sequence of alternating r and s actions. *E.g.* the system $p_1 \stackrel{\text{def}}{=} \mathsf{rec}\mathsf{X}.\big(r.s.X + (a.X + c.\mathbf{0})\big)$ (see [34, Def A.1] for CCS syntax) violates this property since after zero or more serviced queries, p_1 reaches a state that can produce both a and c actions. However, no single trace prefix provides enough evidence to detect this. ■

III. A Framework for Repeated Monitoring

Instrumentation permits the monitor to observe the current execution of the SUS until it detects certain behaviour. We formalise an *extended* online setup, where monitoring is performed in two steps: history aggregation and history analysis. During *aggregation*, monitors gather SUS information over *multiple* executions. Each time a new trace is added to the history, the *analysis* step uses a proof system to determine whether the SUS generating such a history is rejected. If it fails to reject that history, these two steps are repeated until a verdict is reached. SUS instrumentation sits at a lower level of abstraction to the external view used by RECHML which allows monitors to operate with action sequences from TACT.

A. History Aggregation

Monitors. Our runtime analysis, defined in Fig. 2, *records* the traceable actions, TACT, that lead to rejection states. An *executing-monitor* state consists of a tuple (t,m), where t is the trace (*i.e.,* sequence of traceable actions) collected from the beginning of the run, up to the current execution point, and m is the current state of the monitor after analysing it. In order to streamline monitor synthesis from formulae (which only mention external actions) the monitor syntax does not reference internal actions, *e.g.* $\alpha.m$ where $\alpha \in$ EACT in Fig. 2. Accordingly, its monitor semantics determines which *external* actions to record, rules IMON and MACT. Internal actions, used to improve the precision of the history analysis, are recorded by the instrumentation semantics; see rule IASI, discussed later.

Executing-monitor transitions are defined w.r.t. a history H that stores the trace prefixes accumulated in prior executions: $(t,m) \xrightarrow{\eta}_H (t',m')$ denotes that (t,m) transitions to (t',m') either by observing an external action, *i.e.,*$\eta = \alpha$, produced by the SUS or by evolving autonomously via the silent action, *i.e.,* $\eta = \tau$; monitor transitions are never (SUS) internal actions. A monitor execution can reach one of two final states: a *rejection* verdict, no, or an *inconclusive* state, end. The latter behaves like an identity, transitioning to itself when analysing any external SUS action; see rule MEND. Differently, a rejection state indicates to the instrumentation that the (partial) trace analysed thus far should be aggregated to the history. After aggregating the trace, it then behaves as end; see instrumentation rule INO, discussed later. Rule INO is the only rule that extends the history H by the aggregated trace t as (H,t).

The current recorded trace is accrued via monitor sequencing, $\alpha.m$, via rule MACT. Besides sequencing, (sub-)monitors can be composed as a parallel *conjunction*, $m \otimes n$, or *disjunction*, $m \oplus n$. When analysing SUS actions, parallel monitors, $m \odot n$ where $\odot \in \{\oplus, \otimes\}$, move either autonomously, rule MTAUL, or in unison, rule MPAR1. When a sub-monitor cannot analyse the action proffered by the SUS it is discarded (rule MPAR2L); this does not prohibit the former monitor from potentially recording a new trace. An analogous mechanism is also implemented by the instrumentation rule ITER. Four rules determine how a rejection verdict *sub-monitor* is handled. Rule MVRP2L asserts that verdict no supersedes its parallel counterpart whenever the accumulated (violating) trace is new, *i.e.,* $t \notin H$; when $\mathsf{no} \odot n$ transitions to no, it allows the instrumentation rule INO to add t to the history. Dually, if $t \in H$, the rejection verdict is discarded, *i.e.,* $\mathsf{no} \odot n$ transitions to n, to allow n to potentially collect violating traces with common prefixes, rule MVRP1L. The remaining monitor rules are standard, where symmetric rules are elided. Although trace collection does *not* distinguish between parallel conjunction and disjunctions, history analysis does; see Fig. 3.

Instrumentation. The behaviour of an executing-monitor is connected to that of a SUS via the instrumentation relation in Fig. 2. It is defined over *monitored systems*, $H \triangleright (t,m) \triangleleft p$, triples consisting of a SUS p, an executing-monitor (t,m), and a history H. The transition $H \triangleright (t,m) \triangleleft p \xrightarrow{\eta} H' \triangleright (t',m') \triangleleft p'$ denotes that the executing-monitor (t,m) transits to (t',m') when analysing a SUS evolving from p to p' via action η, while updating the history from H to H'. Rule IMON formalises the analysis of an external action, whereas rule INO, previewed earlier, handles the storing of new traces that lead to a rejection verdict. Instrumentation also allows the SUS and executing-monitor to (internally) transition independently of one another, rules IASS and IASM. Rule IASI allows the SUS to transition with an internal action: γ is recorded as part of the aggregated trace while concealing it as a τ action. When (t,m) can neither analyse a SUS action, nor perform an internal transition, the instrumentation forces it to terminate prematurely by transitioning to the inconclusive verdict (rule ITER). This ensures instrumentation transparency [20], [38], where the monitoring infrastructure does not block the behaviour of the SUS whenever the executing monitor cannot analyse an event. We adopt a similar convention to Sec. II; *e.g.* we define weak transitions in a similar manner and write $H \triangleright (t,m) \triangleleft p \stackrel{u}{\Longrightarrow} H' \triangleright (t',m') \triangleleft p'$ in lieu of $H \triangleright (t,m) \triangleleft p \stackrel{\alpha_1}{\Longrightarrow} \cdots \stackrel{\alpha_n}{\Longrightarrow} H' \triangleright (t',m') \triangleleft p'$ for $u = \alpha_1 \cdots \alpha_n$.

Our monitor semantics departs from prior work [8], [11]; it

Monitor Syntax $m,n \in \text{MON} ::= \mathsf{no} \mid \mathsf{end} \mid \alpha.m \mid \mathsf{rec}X.m \mid X \mid m \oplus n \mid m \otimes n \qquad (\odot \in \{\oplus, \otimes\})$

Monitor Semantics

$$\textsc{mEnd}\ \frac{}{(t,\mathsf{end}) \xrightarrow{\alpha}_H (t\alpha,\mathsf{end})} \qquad \textsc{mVrP1L}\ \frac{t \in H}{(t,\mathsf{no} \odot n) \xrightarrow{\tau}_H (t,n)} \qquad \textsc{mVrP2L}\ \frac{t \notin H}{(t,\mathsf{no} \odot n) \xrightarrow{\tau}_H (t,\mathsf{no})} \qquad \textsc{mAct}\ \frac{}{(t,\alpha.m) \xrightarrow{\alpha}_H (t\alpha,m)} \qquad \textsc{mRec}\ \frac{}{(t,\mathsf{rec}X.m) \xrightarrow{\tau}_H (t,m[\mathsf{rec}X.m/X])}$$

$$\textsc{mTauL}\ \frac{(t,m) \xrightarrow{\tau}_H (t,m')}{(t,m \odot n) \xrightarrow{\tau}_H (t,m' \odot n)} \qquad \textsc{mPar1}\ \frac{(t,m) \xrightarrow{\alpha}_H (t',m') \quad (t,n) \xrightarrow{\alpha}_H (t',n')}{(t,m \odot n) \xrightarrow{\alpha}_H (t',m' \odot n')} \qquad \textsc{mPar2L}\ \frac{n \neq \mathsf{no} \quad (t,m) \xrightarrow{\alpha}_H (t',m') \quad (t,n) \not\xrightarrow{\alpha}_H \quad (t,n) \not\xrightarrow{\tau}_H}{(t,m \odot n) \xrightarrow{\alpha}_H (t',m')}$$

Instrumentation Semantics

$$\textsc{iNo}\ \frac{}{H \triangleright (t,\mathsf{no}) \triangleleft p \xrightarrow{\tau} (H,t) \triangleright (t,\mathsf{end}) \triangleleft p} \qquad \textsc{iTer}\ \frac{m \neq \mathsf{no} \quad p \xrightarrow{\alpha} p' \quad (t,m) \not\xrightarrow{\alpha}_H \quad (t,m) \not\xrightarrow{\tau}_H}{H \triangleright (t,m) \triangleleft p \xrightarrow{\alpha} H \triangleright (t\alpha,\mathsf{end}) \triangleleft p'} \qquad \textsc{iAsS}\ \frac{m \neq \mathsf{no} \quad p \xrightarrow{\tau} p'}{H \triangleright (t,m) \triangleleft p \xrightarrow{\tau} H \triangleright (t,m) \triangleleft p'}$$

$$\textsc{iAsI}\ \frac{m \neq \mathsf{no} \quad p \xrightarrow{\gamma} p'}{H \triangleright (t,m) \triangleleft p \xrightarrow{\tau} H \triangleright (t\gamma,m) \triangleleft p'} \qquad \textsc{iAsM}\ \frac{(t,m) \xrightarrow{\tau}_H (t',m')}{H \triangleright (t,m) \triangleleft p \xrightarrow{\tau} H \triangleright (t',m') \triangleleft p} \qquad \textsc{iMon}\ \frac{p \xrightarrow{\alpha} p' \quad (t,m) \xrightarrow{\alpha}_H (t',m')}{H \triangleright (t,m) \triangleleft p \xrightarrow{\alpha} H \triangleright (t',m') \triangleleft p'}$$

Fig. 2. Monitors and Instrumentation

does *not* flag violations but limits itself to aggregating traces. Every monitored execution starts with $t = \varepsilon$ and can, at most, increase the history the current trace accrued. Our monitors work over multiple runs of the *same* SUS. Starting from an empty history $H_0{=}\emptyset$, traces leading to no states, can be accumulated over a sequence of monitored SUS executions by passing history H_i obtained from the i^{th} monitored execution on to execution $i{+}1$, inducing a (finite) totally-ordered sequence of histories, $\emptyset{=}H_0{\subseteq}H_1{\subseteq}\cdots$

Example III.1. Monitor $m_1 \stackrel{\text{def}}{=} \mathsf{rec}X.\big(r.s.X \otimes (a.\mathsf{no} \oplus c.\mathsf{no})\big)$ reaches state no after observing actions a or c, following a sequence of serviced queries. System $p_2 \stackrel{\text{def}}{=} \mathsf{rec}X.\big(r.s.X + (\delta_1.a.X + \delta_2.c.\mathbf{0})\big)$ extends p_1 from Ex. II.1, where the decision on whether to allocate memory or close depends on checking whether there is free memory or not, expressed as the internal actions δ_1 and δ_2 respectively. When p_2 is instrumented with the executing-monitor (ε, m_1) and history $H_0 = \emptyset$, it can reach state no through the prefix $t_1 = rs\delta_1 a$ as shown below. With the augmented history $H_1 = \{t_1\}$, the monitored SUS $H_1 \triangleright (\varepsilon, m_1) \triangleleft p_2$ can then aggregate $t_2 = rs\delta_2 c$ in a subsequent run, *i.e.,* $H_1 \triangleright (\varepsilon, m_1) \triangleleft p_2 \stackrel{t_2}{\Longrightarrow} H_2 \triangleright (t_2, \mathsf{end}) \triangleleft p_2$ where $H_2 = \{t_1, t_2\}$.

$$\begin{aligned}
&H_0 \triangleright (\varepsilon, m_1) \triangleleft p_2 \xrightarrow{\tau} \cdot \xrightarrow{\tau} && \text{(IAsS,IAsM)}\\
&\quad H_0 \triangleright (\varepsilon, r.s.m_1 \otimes (a.\mathsf{no} \oplus c.\mathsf{no})) \triangleleft r.s.p_2 + (\delta_1.a.p_2 + \delta_2.c.\mathbf{0})\\
&\xrightarrow{r} H_0 \triangleright (r, s.m_1) \triangleleft s.p_2 \xrightarrow{s} H_0 \triangleright (rs, m_1) \triangleleft p_2 && \text{(iMon)}\\
&\xrightarrow{\tau} \cdot \xrightarrow{\tau} && \text{(iAsP, iAsM)}\\
&\quad H_0 \triangleright (rs, r.s.m_1 \otimes (a.\mathsf{no} \oplus c.\mathsf{no})) \triangleleft r.s.p_2 + (\delta_1.a.p_2 + \delta_2.c.\mathbf{0})\\
&\xrightarrow{\tau} H_0 \triangleright (rs\delta_1, r.s.m_1 \otimes (a.\mathsf{no} \oplus c.\mathsf{no})) \triangleleft a.p_2 && \text{(iAsI)}\\
&\xrightarrow{a} H_0 \triangleright (rs\delta_1 a, \mathsf{no}) \triangleleft p_2 && \text{(iMon)}\\
&\xrightarrow{\tau} H_0 \cup \{rs\delta_1 a\} \triangleright (rs\delta_1 a, \mathsf{end}) \triangleleft p_2 && \text{(iNo)}
\end{aligned}$$

Note that, since monitors assume a passive role [20], they cannot steer the behaviour of the SUS, meaning the SUS may *not* exhibit *different* behaviour across multiple executions. ■

The instrumentation mechanism needs to aggregate overlapping trace prefixes that lead to rejection states.

Example III.2. The SUS p_2 from Ex. III.1 generates traces of the form $(rs\delta_1 a)^*$. Monitor $m_2 \stackrel{\text{def}}{=} \mathsf{rec}X.\big(r.s.X \otimes a.X \otimes (a.\mathsf{no} \oplus c.\mathsf{no})\big)$ revises m_1 where sequences of rs actions can be interleaved with finite sequences of a actions described by the sub-monitor $a.X$. When (ε, m_2) is instrumented on p_2 with $H_0 = \emptyset$, it can record the prefix $rs\delta_1 a$ during a first run. In a subsequent run with an augmented $H_1{=}\{rs\delta_1 a\}$, we have:

$$\begin{aligned}
&H_1 \triangleright (\varepsilon, m_2) \triangleleft p_2\\
&\stackrel{rs\delta_1}{\Longrightarrow} H_1 \triangleright (rs\delta_1, r.s.m_2 \otimes a.m_2 \otimes (a.\mathsf{no} \oplus c.\mathsf{no})) \triangleleft a.p_2\\
&\xrightarrow{a} H_1 \triangleright (rs\delta_1 a, m_2 \otimes \mathsf{no}) \triangleleft p_2 \xrightarrow{\tau} H_1 \triangleright (rs\delta_1 a, m_2) \triangleleft p_2 \quad (*)\\
&\stackrel{rs\delta_1 a}{\Longrightarrow} H_1 \triangleright (rs\delta_1 ars\delta_1 a, m_2 \otimes \mathsf{no}) \triangleleft p_2\\
&\xrightarrow{\tau} H_1 \triangleright (rs\delta_1 ars\delta_1 a, \mathsf{no}) \triangleleft p_2\\
&\xrightarrow{\tau} H_1 \cup \{rs\delta_1 ars\delta_1 a\} \triangleright (rs\delta_1 ars\delta_1 a, \mathsf{end}) \triangleleft p_2 \quad (\dagger)
\end{aligned}$$

Transition $(*)$ follows rule MVRP1R with $(rs\delta_1 a, m_2 \otimes \mathsf{no}) \xrightarrow{\tau}_{H_1} (rs\delta_1 a, m_2)$ since $rs\delta_1 a \in H_1$: the executing-monitor does *not* stop accruing at $rs\delta_1 a$ but continues monitoring until it encounters a *new* rejecting trace, $rs\delta_1 ars\delta_1 a$, which is aggregated to H_1 in transition $(\dagger)$ using rule INO. ■

Remark 2. Rule INO encodes the design decision to stop monitoring (by transitioning to end) as soon as a new trace is aggregated to the history, providing a clear cut-off point for when to pass the aggregated history to the subsequent run. ■

B. History Analysis

We formalise how a history is rejected by a monitor through a proof system. Its main judgement is $\mathbf{rej}_{\text{DET}}(H, \mathsf{f}, m)$, *i.e.,* monitor m *rejects* history H using DET with the boolean flag f. It can be seen as an attempt to reconstruct a partial model of the SUS from the traces aggregated, large enough to infer the

$$\text{NO}\ \frac{H \neq \emptyset}{\mathbf{rej}_{\text{DET}}(H,\mathsf{f},\mathsf{no})} \qquad \text{ACT}\ \frac{H' = sub(H,\alpha) \quad \mathbf{rej}_{\text{DET}}(H',(\mathsf{f} \wedge \text{DET}(\alpha)),m)}{\mathbf{rej}_{\text{DET}}(H,\mathsf{f},\alpha.m)}$$

$$\text{ACTI}\ \frac{H' = sub(H,\gamma) \quad \mathbf{rej}_{\text{DET}}(H',(\mathsf{f} \wedge \text{DET}(\gamma)),\alpha.m)}{\mathbf{rej}_{\text{DET}}(H,\mathsf{f},\alpha.m)} \qquad \text{PARAL}\ \frac{\mathbf{rej}_{\text{DET}}(H,\mathsf{f},m)}{\mathbf{rej}_{\text{DET}}(H,\mathsf{f},m \otimes n)}$$

$$\text{PARO}\ \frac{\mathbf{rej}_{\text{DET}}(H,\mathsf{true},m) \quad \mathbf{rej}_{\text{DET}}(H,\mathsf{true},n)}{\mathbf{rej}_{\text{DET}}(H,\mathsf{true},m \oplus n)} \qquad \text{REC}\ \frac{\mathbf{rej}_{\text{DET}}(H,\mathsf{f},m[\mathsf{rec}\,\mathsf{X}.m/\mathsf{X}])}{\mathbf{rej}_{\text{DET}}(H,\mathsf{f},\mathsf{rec}\,\mathsf{X}.m)}$$

Fig. 3. Proof System

violation. This judgement uses internal actions and DET to calculate whether the traces are produced by the same states (up to $\equiv$); the flag value true encodes that all the actions analysed up to this point were deterministic actions. This analysis is the least relation defined by the rules in Fig. 3, relying on a helper function $sub(H,\mu) = \{t \mid \mu t \in H\}$; it returns the continuation of any trace in H that is prefixed by a μ action; *e.g.* when $H = \{rsa, rsc, ars\}$, we get $sub(H,r) = \{sa, sc\}$. The axiom NO states that a no monitor rejects all *non-empty* histories, *i.e.,* a monitor cannot reject a SUS outright, without any observation. In rule ACT, a sequenced monitor $\alpha.m$ rejects H with flag f if the (sub-)monitor m rejects the history returned by $sub(H,\alpha)$ with updated flag $(\mathsf{f} \wedge \text{DET}(\alpha))$. Alternatively, $\alpha.m$ can reject H with f following rule ACTI, by considering the suffixes of traces prefixed by an internal action γ, again updating the flag to $(\mathsf{f} \wedge \text{DET}(\gamma))$. Parallel conjunctions $m \otimes n$ reject H with f if either *one* of the constituent monitors m and n rejects H with f (rules PARAL and PARAR). Importantly, parallel disjunctions $m \oplus n$ reject H with only when the flag is true and *both* monitors reject it (rule PARO), ensuring that the trace prefix analysed consisted of deterministic actions. Rule REC states that a recursive monitor rejects a history with some flag if its unfolding does. As a shorthand, we say that monitor m rejects history H, denoted $\mathbf{rej}_{\text{DET}}(H,m)$, whenever $\mathbf{rej}_{\text{DET}}(H,\mathsf{true},m)$.

Example III.3. Recall p_2 and m_1 from Ex. III.1 and suppose that $\text{DET}(r) = \text{DET}(s) = \mathsf{true}$. Instrumentation can record $t_1 = rs\delta_1 a$ during a first execution, but m_1 fails to reject the recorded history, $\neg\mathbf{rej}_{\text{DET}}(\{t_1\},m_1)$. When p_2 is monitored again, the additional trace $t_2 = rs\delta_2 c$ can be aggregated, which m_1 now rejects, $\mathbf{rej}_{\text{DET}}(\{t_1,t_2\},m_1)$ (see [34, Figs. 4 and 5]). ■

Ex. III.4 shows that rejections are always evidence-based.

Example III.4. Although monitor no trivially rejects *any* p, it does so after observing *one* execution: for $H_0 = \emptyset$, the semantics in Fig. 2 immediately triggers rule INO, *i.e.,* $\emptyset \triangleright (\varepsilon,\mathsf{no}) \triangleleft p \xrightarrow{\tau} \{\varepsilon\} \triangleright (\varepsilon,\mathsf{end}) \triangleleft p$. When ε is added to the history, one can conclude $\mathbf{rej}_{\text{DET}}(\{\varepsilon\},\mathsf{no})$ by rule NO. ■

IV. MONITOR CORRECTNESS

RV establishes a correspondence between the operational behaviour of a monitor and the semantic meaning of the property being monitored for [39], [23] which transpires the meaning of the statement "monitor *m correctly monitors* for a property φ." Our first correctness result concerns the *history aggregation* mechanism of Sec. III. Prop. IV.1 states that traces collected are indeed generated by the instrumented SUS. Thus, whenever a history H is accumulated over a sequence of executions of some p, *i.e.,* $\emptyset \subseteq H_1 \subseteq \cdots \subseteq H$, then $H \subseteq T_p$.

Proposition IV.1 (Veracity)**.** *For any H, m, p, and* $\eta_1,\ldots,\eta_n$, *if* $H \triangleright (\varepsilon,m) \triangleleft p \xrightarrow{\eta_1} \ldots \xrightarrow{\eta_n} H' \triangleright (t,m') \triangleleft p'$ *then* $p \overset{t}{\Longrightarrow}_T p'$. ■

Another criteria for our multi-run monitoring setup is that executing-monitors *behave deterministically* [38], [40]. Our monitors are *confluent* w.r.t. τ-moves [34, Prop. C.6], thus monitors are equated up to τ-transitions. Importantly, for a given history, the executing-monitors of Sec. III deterministically reach equivalent states when analysing a (partial) trace exhibited by the SUS, Prop. IV.2.

Proposition IV.2 (Determinism)**.** *If* $(t,m) \overset{u}{\underset{H}{\Longrightarrow}} (t',m')$ *and* $(t,m) \overset{u}{\underset{H}{\Longrightarrow}} (t'',m'')$, *then* $t' = t''$ *and there is* $n \in \text{MON}$ *such that* $(t',m')(\xrightarrow{\tau}_H)^*(t',n)$ *and* $(t'',m'')(\xrightarrow{\tau}_H)^*(t'',n)$. ■

Example IV.1. Recall m_2 from Ex. III.2. Given $u = rsa$, the executing-monitor (ε,m_2) can reach either (u,no) or $(u,m \otimes \mathsf{no})$ which τ-converges to (u,no) via rule MVRP2R. ■

A characteristic sanity check is *verdict irrevocability* [20], [38], [23]. This translates to Prop. IV.3 stating that, once a SUS is rejected for exhibiting history H (using the *history analysis* of Fig. 3), further observations (in terms of longer traces, *length*, or additional traces, *width*) do *not* alter this conclusion.

Proposition IV.3 (Irrevocability)**.**
Length: *If* $\mathbf{rej}_{\text{DET}}((H,t),m)$ *then* $\mathbf{rej}_{\text{DET}}((H,tu),m)$.
Width: *If* $\mathbf{rej}_{\text{DET}}(H,m)$ *then* $\mathbf{rej}_{\text{DET}}(H \cup H',m)$. ■

The least *correctness* requirement expected of our (irrevocable) *history analysis* is that any rejections imply property violations. Concretely, m monitors soundly for φ if, for *any* system p, whenever m rejects a history H produced by p, *i.e.,* $\mathbf{rej}_{\text{DET}}(H,m)$ for $H \subseteq T_p$, then p also violates the property, *i.e.,* $p \notin [\![\varphi]\!]$. The universal quantification over systems required by Def. IV.1 manifests a black-box treatment of the SUS.

Definition IV.1 (Soundness)**.** m monitors *soundly* for φ when $\forall p \in \text{PRC}$, if $\exists H \subseteq T_p$ such that $\mathbf{rej}_{\text{DET}}(H,m)$ then $p \notin [\![\varphi]\!]$. ■

Example IV.2. m_1 from Ex. III.1 monitors soundly for φ_4 from Ex. II.1. Specifically, Ex. III.1 illustrates how trace prefixes $rs\delta_1 a$ and $rs\delta_2 c$ of p_2 can be veraciously accumulated as a history and Ex. III.3 shows that such a history is rejected. Accordingly, p_2 violates φ_4. By comparison, monitor $m_3 \overset{\text{def}}{=} r.s.a.\mathsf{no}$ is *not* sound for φ_4; it can collect and reject histories that contain the trace $rs\delta_1 a$, but systems such as $\mathsf{rec}\,X.r.s.\delta_1.a.X$ and $r.s.\delta_1.a.\mathbf{0}$ (which can exhibit such a trace) satisfy φ_4, *i.e.,* they do *not* violate it. ■

The dual requirement to soundness is (rejection) completeness: *m monitors completely* for φ if any $p \notin [\![\varphi]\!]$ can be rejected based on some history it produces.

Definition IV.2 (Completeness)**.** m monitors *completely* for φ when $\forall p \in \text{PRC}$, if $p \notin [\![\varphi]\!]$ then $\exists H \subseteq T_p$ such that $\mathbf{rej}_{\text{DET}}(H,m)$. ■

Example IV.3. It can be shown that $m_4 \stackrel{\text{def}}{=} s.\mathsf{no} \otimes a.\mathsf{no} \otimes c.\mathsf{no}$ monitors completely for φ_0 from Ex. I.1. Concretely, any violating system can exhibit a trace of the form ts, ta or tc for some $t \in \text{IACT}^*$. Once exhibited (and aggregated), one can show that m_4 rejects such a history. ■

For monitors that are veracious and produce irrevocable verdicts (Sec. III), (rejection) soundness and completeness constitute the basis for our definition of monitor correctness.

Definition IV.3 (Correct Monitoring)**.** A monitor m monitors *correctly* for the (closed) formula φ if it can do so *soundly* and *completely*. ■

V. MONITORABILITY

Monitorability [41], [8], [21], [23] delineates the properties that can be correctly monitored from those that cannot, formalised as a correspondence between the declarative logic semantic of Sec. II and the operational monitor semantics of Sec. III. The chosen approach [39] applies to a variety of settings [4], [11], [42], [43], [44], [45], [46]. It fosters a separation of concerns between the specification semantics and the verification method employed, which is relevant to our investigation on the increase in expressive power when moving from single-run monitoring to multi-runs; see [23] for a comparison between distinct notions of monitorability. Specifically, Def. V.1 (below) is *parametric* w.r.t. the definition of *"m monitors correctly for φ"*; prior work [8] formalised this as single-run monitoring whereas Def. IV.3 redefines it as multi-run monitoring.

Definition V.1 (Monitorability [8])**.** Formula $\varphi \in \text{RECHML}$ is *monitorable* iff $\exists m \in \text{MON}$ monitoring *correctly* for it. Sublogic $\mathscr{L} \subseteq \text{RECHML}$ is monitorable iff $\forall \varphi \in \mathscr{L}$ are monitorable. ■

Several formulae are unmonitorable (for violations) according to Def. V.1, particularly when they include existential modalities and least fixed points.

Example V.1. Assume, towards a contradiction, that there exists a sound and complete monitor m for the formula $\langle\alpha\rangle\mathsf{tt}$. Pick some $p \notin [\![\langle\alpha\rangle\mathsf{tt}]\!]$, *i.e.,* $p \stackrel{\alpha}{\not\rightarrow}$. By the completeness requirement of Def. IV.2, there exists a history $H \subseteq T_p$ such that $\mathbf{rej}_{\text{DET}}(H,m)$. Irrespective of the value of $\text{DET}(\alpha)$, we can use p to build another system $p+\alpha.\mathbf{0}$ where $p+\alpha.\mathbf{0} \in [\![\langle\alpha\rangle\mathsf{tt}]\!]$. We also know that H is a history of $p+\alpha.\mathbf{0}$ since $H \subseteq T_p \subseteq T_{(p+\alpha.\mathbf{0})}$. This fact and $\mathbf{rej}_{\text{DET}}(H,m)$ makes m unsound, contradicting our initial assumption.

Similarly, assume, towards a contradiction, that there exists a monitor m that can monitor soundly and completely for $\min\mathsf{X}.([\alpha]\mathsf{X} \wedge [\beta]\mathsf{ff})$. The single-state system p with the sole transition $p \xrightarrow{\alpha} p$ violates the formula. Due to completeness of Def. IV.2, we must have $\mathbf{rej}_{\text{DET}}(H,m)$ for some $H \subseteq T_p$. From the structure of p, we also know H is a *finite* set of the form $\{\alpha^n \mid n \in \mathbb{N}\}$. Fix k to be the length of the *longest* trace in H and then consider the system q consisting of $k+1$ states that exclusively has the transitions $q = q_0 \xrightarrow{\alpha} \ldots \xrightarrow{\alpha} q_k$ (and nothing else). Clearly, q satisfies $\min\mathsf{X}.([\alpha]\mathsf{X} \wedge [\beta]\mathsf{ff})$. Since $H \subseteq T_q$ as well, $\mathbf{rej}_{\text{DET}}(H,m)$ contradicts the initial assumption that m is sound (and complete). ■

Disjunctions are the only other RECHML logical constructs excluded from SHML, as restated in Thm. II.2. Formulae containing disjunctions can be monitorable with a few caveats.

Example V.2. Recall $\varphi_2 \stackrel{\text{def}}{=} [r]([s]\mathsf{ff} \vee [a]\mathsf{ff})$ from Ex. I.3. When $\text{DET}(r) = \mathsf{false}$, φ_2 is *not* monitorable. By contradiction, assume a correct m exists. Since $p_3 \stackrel{\text{def}}{=} r.(s.\mathbf{0}+a.\mathbf{0})+r.s.\mathbf{0} \notin [\![\varphi_2]\!]$, then we should have $\mathbf{rej}_{\text{DET}}(H,m)$ for some $H \subseteq T_{p_3}$. But $H \subseteq T_{p_4}=T_{p_3}$ for $p_4 \stackrel{\text{def}}{=} r.s.\mathbf{0}+r.a.\mathbf{0} \in [\![\varphi_2]\!]$, and $\mathbf{rej}_{\text{DET}}(H,m)$ would make m unsound, contradicting our initial assumption.

However, when $\text{DET}(r)=\mathsf{true}$, φ_2 is monitorable: an obvious correct monitor is $m_5 \stackrel{\text{def}}{=} r.(s.\mathsf{no} \oplus a.\mathsf{no})$. Although systems p_3 and p_4 would be ruled out by $\text{DET}(r)=\mathsf{true}$, an ILTS would still allow systems such as $p_5 \stackrel{\text{def}}{=} r.(s.\mathbf{0}+a.\mathbf{0})+r.(s.\mathbf{0}+a.\mathbf{0}+a.\mathbf{0})$ that reaches the equivalent states $s.\mathbf{0}+a.\mathbf{0}$ and $s.\mathbf{0}+a.\mathbf{0}+a.\mathbf{0}$ after an r-transition. Even if $H = \{ra, rs\}$ is aggregated by passing through *different* intermediary states, *i.e.,* $s.\mathbf{0}+a.\mathbf{0}$ and $s.\mathbf{0}+a.\mathbf{0}+a.\mathbf{0}$, the monitor analysis would still be sound in rejecting p_5 via H; see Prop. II.1.

A trickier formula is $\varphi_4 \stackrel{\text{def}}{=} \max\mathsf{X}.\big([r][s]\mathsf{X} \wedge ([a]\mathsf{ff} \vee [c]\mathsf{ff})\big)$ from Ex. II.1. Although the disjunction is syntactically not prefixed by any universal modality, it can be reached after a recursive unfolding, *i.e.,* $\varphi_4 \equiv [r][s]\varphi_4 \wedge ([a]\mathsf{ff} \vee [c]\mathsf{ff})$. By similar reasoning to that for φ_2, formula φ_4 is monitorable whenever $\text{DET}(r)=\text{DET}(s)=\mathsf{true}$ but unmonitorable otherwise. ■

Def. V.2 characterises the extended class of RECHML monitorable formulae for multi-run monitoring, parametrised by EACT and the associated action determinacy delineation defined by DET. It employs a flag to calculate deterministic prefixes via rule CUM along the lines of Fig. 3. This is then used by rule COR, which is only defined when the flag is true.

Definition V.2. $\mathsf{f} \vdash_{\text{DET}} \varphi$ is defined coinductively as the largest relation of the form $(\text{BOOL} \times \text{RECHML})$ satisfying the rules

$$\text{CA}\ \frac{\varphi \in \{\mathsf{ff},\mathsf{tt},\mathsf{X}\}}{\mathsf{f} \vdash_{\text{DET}} \varphi} \qquad \text{CUM}\ \frac{\mathsf{f} \wedge \text{DET}(\alpha) \vdash_{\text{DET}} \varphi}{\mathsf{f} \vdash_{\text{DET}} [\alpha]\varphi} \qquad \text{CAND}\ \frac{\mathsf{f} \vdash_{\text{DET}} \varphi \quad \mathsf{f} \vdash_{\text{DET}} \psi}{\mathsf{f} \vdash_{\text{DET}} \varphi \wedge \psi}$$

$$\text{COR}\ \frac{\mathsf{true} \vdash_{\text{DET}} \varphi \quad \mathsf{true} \vdash_{\text{DET}} \psi}{\mathsf{true} \vdash_{\text{DET}} \varphi \vee \psi} \qquad \text{CMAX}\ \frac{\mathsf{f} \vdash_{\text{DET}} \varphi[\max\mathsf{X}.\varphi/\mathsf{X}]}{\mathsf{f} \vdash_{\text{DET}} \max\mathsf{X}.\varphi}$$

$\text{SHML}^{\vee}_{\text{DET}} \stackrel{\text{def}}{=} \{\, \varphi \mid \mathsf{true} \vdash_{\text{DET}} \varphi \,\}$ defines the set of extended monitorable formulae. It extends SHML with disjunctions as long as these are prefixed by universal modalities of deterministic external actions (up to largest fixed point unfolding). ■

Example V.3. Assuming that $\text{DET}(r) = \text{DET}(s) = \mathsf{true}$ and that $\text{DET}(a) = \mathsf{false}$, we can symbolically show that both formulae φ_2 and φ_4 are in $\text{SHML}^{\vee}_{\text{DET}}$. For instance, recall that $\varphi_2 = [r]([s]\mathsf{ff} \vee [a]\mathsf{ff})$. According to Def. V.2, to justify the inclusion $\varphi_2 \in \text{SHML}^{\vee}_{\text{DET}}$

it suffices to prove the judgement $\mathsf{true} \vdash_{\text{DET}} \varphi_2$. Now, the relation $R = \{(\mathsf{true}, [r]([s]\mathsf{ff} \vee [a]\mathsf{ff})), (\mathsf{true}, [s]\mathsf{ff} \vee [a]\mathsf{ff}), (\mathsf{true}, [s]\mathsf{ff}), (\mathsf{true}, [a]\mathsf{ff}), (\mathsf{true}, \mathsf{ff}), (\mathsf{false}, \mathsf{ff})\}$ satisfies the coinductive rules of Def. V.2 and includes the pair $(\mathsf{true}, \varphi_2)$, thereby proving the judgement $\mathsf{true} \vdash_{\text{DET}} \varphi_2$. ■

Although the tracing of internal actions as part of the history helps with correct monitoring, multi-run RV requires us to limit systems to deterministic internal actions in order to attain violation completeness for monitors MON of Fig. 2.

Example V.4. The two systems $p_6 \stackrel{\text{def}}{=} \delta_1.r.s.\mathbf{0} + \delta_2.r.a.\mathbf{0}$ and $p_7 \stackrel{\text{def}}{=} \gamma.r.s.\mathbf{0} + \gamma.r.a.\mathbf{0}$ both *satisfy* φ_2 from Ex. V.2 when $\text{DET}(r) = \mathsf{true}$. In the case of p_6, the correct monitor m_5 from Ex. V.2 does *not* reject the history $\{\delta_1 rs, \delta_2 ra\}$ because the application of rule ACTI of Fig. 3 (for either δ_1 or δ_2) necessarily reduces the history size of the premise to *one* trace. For system p_7, we must also have $\text{DET}(\gamma) = \mathsf{false}$; when m_5 analyses the history $\{\gamma rs, \gamma ra\}$ using rule ACTI, the premise flag can only be false which prohibits the analysis from using PARO. Both systems $p_8 \stackrel{\text{def}}{=} r.(\delta_1.s.\mathbf{0} + \delta_2.a.\mathbf{0})$ and $p_9 \stackrel{\text{def}}{=} r.(\gamma.s.\mathbf{0} + \gamma.a.\mathbf{0})$ *violate* φ_2. Accordingly, both are rejected by m_5 via the respective histories $\{r\delta_1 s, r\delta_2 a\}$ and $\{r\gamma s, r\gamma a\}$.

Non-deterministic internal actions hinder completeness. System $p_{10} \stackrel{\text{def}}{=} \gamma.p_8 + \gamma.\mathbf{0}$ violates φ_2 but m_5 cannot reject the history $\{\gamma r\delta_1 s, \gamma r\delta_2 a\}$: again, $\text{DET}(\gamma) = \mathsf{false}$ limits the flag premises for ACTI to false, prohibiting the use of PARO. ■

Showing that a logical fragment, $\mathscr{L} \subseteq$ RECHML, is monitorable, Def. V.1, can be onerous due to the various universal quantifications that need to be considered, *e.g.* all formulae $\varphi \in \mathscr{L}$, all systems $p \in$ PRC and all histories $H \in$ HST from Defs. IV.1 and IV.2. We prove the monitorability of $\text{sHML}^{\vee}_{\text{DET}}$ systematically, by concretising the existential quantification of a correct monitor for every $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$ via the monitor synthesis $(\!|-|\!)$. We then prove that for any $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$, the synthesised monitor $(\!|\varphi|\!)$ does monitor correctly for it (Def. V.1). A by-product of this proof strategy is that the synthesis function in Def. V.3 can be used directly for tool construction to automatically generate (correct) witness monitors from specifications; see [12], [47].

Definition V.3. $(\!|-|\!) : \text{sHML}^{\vee}_{\text{DET}} \to \text{MON}$ is defined as follows:

$$(\!|\mathsf{ff}|\!) \stackrel{\text{def}}{=} \mathsf{no} \quad (\!|\varphi \wedge \varphi|\!) \stackrel{\text{def}}{=} (\!|\varphi|\!) \otimes (\!|\varphi|\!) \quad (\!|[\alpha]\varphi|\!) \stackrel{\text{def}}{=} \alpha.(\!|\varphi|\!) \quad (\!|\mathsf{X}|\!) \stackrel{\text{def}}{=} \mathsf{X}$$
$$(\!|\mathsf{tt}|\!) \stackrel{\text{def}}{=} \mathsf{end} \quad (\!|\varphi \vee \varphi|\!) \stackrel{\text{def}}{=} (\!|\varphi|\!) \oplus (\!|\varphi|\!) \quad (\!|\max \mathsf{X}.\varphi|\!) \stackrel{\text{def}}{=} \mathsf{rec}\mathsf{X}.(\!|\varphi|\!) \quad ■$$

If we limit ILTSs to deterministic internal actions, *i.e.,* $\text{DET}(\gamma) = \mathsf{true}$ for all $\gamma \in$ IACT, we can show monitorability for arbitrary ILTSs and the fragment $\text{sHML}^{\vee}_{\text{DET}}$.

Proposition V.1. $(\!|\varphi|\!)$ *is sound for* $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$. ■

Proposition V.2. *If* $\text{DET}(\gamma) = \mathsf{true}$ *for all* $\gamma \in$ IACT*, then* $(\!|\varphi|\!)$ *is complete for all* $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$. ■

Theorem V.3 (Monitorability)**.** *When* $\text{DET}(\gamma) = \mathsf{true}$ *for all* $\gamma \in$ IACT*, all* $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$ *are monitorable.* ■

We can show an even stronger result called *maximality* which ensures that restricting specifications to $\text{sHML}^{\vee}_{\text{DET}}$ does not exclude any monitorable properties, Thm. V.4. Maximality typically relies on a reverse synthesis $\langle\langle - \rangle\rangle$ that maps any monitor $m \in$ MON to a characteristic formula $\langle\langle m \rangle\rangle \in \text{sHML}^{\vee}_{\text{DET}}$ that it monitors correctly for. This method is however complicated by the occurrence of non-deterministic actions. For instance, if $\text{DET}(r) = \mathsf{false}$, the monitor $r.(s.\mathsf{no} \oplus a.\mathsf{no})$ does *not* correctly monitor for $[r]([s]\mathsf{ff} \vee [a]\mathsf{ff})$ but instead never rejects. In order to overcome these anomalies and obtain our results, we first normalise the aforementioned monitor to $r.\mathsf{end}$; see [34, Sec. E].

Maximality permits a verification framework to determine if a property is monitorable via a simple syntactic check, or else employ alternative verification techniques. The development of an RV tool can also exclusively target $\text{sHML}^{\vee}_{\text{DET}}$, knowing that all monitorable properties are covered.

Theorem V.4 (Maximality)**.** *If* $\text{DET}(\gamma) = \mathsf{true}$ *for all* $\gamma \in$ IACT *and* $\mathscr{L} \subseteq$ RECHML *is monitorable w.r.t.* MON*, then for all* $\varphi \in \mathscr{L}$*, there exists* $\psi \in \text{sHML}^{\vee}_{\text{DET}}$ *such that* $[\![\varphi]\!] = [\![\psi]\!]$. ■

VI. ACTOR SYSTEMS

We validate the utility and applicability of monitoring ILTSs from Sec. II via an instantiation to actor systems [48], [49], [50], [51], [52], [53] where a set of concurrent processes called *actors* interact via *asynchronous message-passing*. Each actor, $i[e \triangleleft q]$, is identified by its unique ID, $i, j, h, k \in$ PID, used by other actors (possibly more than one) to address messages to it *i.e.,* the *single-receiver* property. Internally, actors consist of a running expression e and a mailbox q, *i.e.,* a list of values denoting a message queue.

$$A, B \in \text{ACTR} ::= \; i[e \triangleleft q] \;\mid\; \mathbf{0} \;\mid\; A \parallel B \;\mid\; (\nu\, i)A \;\mid\; i\langle v \rangle$$

Parallel actors, $A \parallel B$, can also be inactive, $\mathbf{0}$, or have IDs that are locally *scoped* to a subset of actors, $(\nu\, i)A$. There may also be asynchronous messages in transit, $i\langle v \rangle$, where value v is addressed to i. The set of all free IDs i identifying actors $i[e \triangleleft q]$ in A is denoted by $\mathit{fId}(A)$.

Values, $v \in$ VAL, range over PID $\cup$ ATOM where $a, b \in$ ATOM are uninterpreted tags. Actor expressions $e, d \in$ EXP can be outputs, $i!v.e$, or reading inputs from the mailbox through pattern-matching, $\mathsf{rcv}\{p_n \to e_n\}_{n \in I}$, where each expression e_n is guarded by a *disjoint* pattern p_n. Actors may also refer to themselves, $\mathsf{self}\, x.e$, spawn other actors, $\mathsf{spw}\, d\, \mathsf{as}\, x.e$, or recurse, $\mathsf{rec}\mathsf{X}.e$. Receive patterns, spawn and recursion bind expression variables $x, y \in$ VARS, and term variables $\mathsf{X}, \mathsf{Y} \in$ TVAR. Similarly, $(\nu\, i)A$ binds the name ID i in A. We work up to α-conversion of bound entities. The list notation $v : q$ denotes the mailbox with v as the head and q as the tail of the queue, whereas $q : v$ denotes the mailbox with v at the end of the queue preceded by q; queue concatenation is denoted as $q : r$. We elide empty mailboxes, ε, and write $i[e]$ for $i[e \triangleleft \varepsilon]$.

The ILTS semantics for our language is defined over system states of the form $K \mid O \triangleright A \in$ PRC. The implicit *observers* that A interacts with when running is represented by the set of

IDs $O \subseteq \text{PID}$; to model the single receiver property we have $fId(A) \cap O = \emptyset$. *Knowledge*, $K \subseteq \text{PID}$, denotes the set of IDs known to all actors in A and O; it keeps track of bound/free names without the need for name bindings in actions [54] where $(fId(A) \cup O) \subseteq K$; see [55]. Transitions are of the form

$$K \mid O \triangleright A \xrightarrow{\eta} K' \mid O' \triangleright B \tag{1}$$

where η ranges over $\text{EACT} \cup \text{IACT} \cup \{\tau\}$. *External* actions $\text{EACT} = \{ i?v,\ i!v,\ i\uparrow j \mid i,j \in \text{PID}, v \in \text{VAL} \}$ include input, $i?v$, output, $i!v$, and scope-extruding output, $i\uparrow j$. *Internal* actions $\text{IACT} = \{ \mathsf{com}(i,v),\ \mathsf{ncom} \mid i \in \text{PID}, v \in \text{VAL} \}$ include internal communication involving either free names, $\mathsf{com}(i,v)$ or scoped names, ncom. Eq. (1) is governed by the judgement $K \mid O \triangleright A \xrightarrow{\eta} B$ with $K'|O' = aft(K|O,\eta)$; the latter function determines K and O where $aft(K|O, i\uparrow j) \stackrel{\text{def}}{=} \big(K \cup \{j\}\big)|O$ and $aft(K|O, i?j) \stackrel{\text{def}}{=} (K \cup \{j\})|\big(O \cup (\{j\} \setminus K)\big)$ (all other cases of η leave $K|O$ unchanged). The generation of *external actions* is defined by the following rules where asynchronous output is conducted in two steps, rules SND1 and SND2, where the latter rule requires the recipient address j to be in O. Scope-extruded outputs with its name management is described by OPN.

$$\text{SND1}\ \frac{}{K \mid O \triangleright i[j!v.e \triangleleft q] \xrightarrow{\tau} i[e \triangleleft q] \parallel j\langle v \rangle} \qquad \text{SND2}\ \frac{}{K \mid O \triangleright j\langle v \rangle \xrightarrow{j!v} \mathbf{0}}\ j \in O$$

$$\text{RCV}\ \frac{}{K \mid O \triangleright i[e \triangleleft q] \xrightarrow{i?v} i[e \triangleleft q : v]} \qquad \text{OPN}\ \frac{(K, j) \mid O \triangleright A \xrightarrow{i!j} B}{K \mid O \triangleright (\nu\, j)A \xrightarrow{i\uparrow j} B}$$

$$\text{RD}\ \frac{\forall n \in I \cdot absent(p_n, q) \quad \exists m \in I \cdot \neg\, absent(p_m, v) \wedge match(p_m, v) = \sigma}{K \mid O \triangleright i[\mathsf{rcv}\,\{p_n \to e_n\}_{n \in I} \triangleleft q : v : r] \xrightarrow{\tau} i[e_m\sigma \triangleleft q : r]}$$

Rule RCV details how input actions append to the recipient mailbox, which are then *selectively* read following rule RD. Selection relies on the helper functions $absent(-)$ and $match(-)$ in [34, Def. H.1] to find the first message v in the mailbox that matches one of the patterns p_m in $\{p_n \to e_n\}_{n \in I}$. If a match is found, the actor branches to $e_m\sigma$, where e_m is the expression guarded by the matching pattern p_m and $\sigma \in \text{SUB} : \text{VARS} \rightharpoonup \text{VAL}$ substitutes the free variables in e_m for the values resulting from the pattern-match.

$$\text{COMML}\ \frac{K \mid fId(B) \triangleright A \xrightarrow{i!v} A' \quad K \mid fId(A) \triangleright B \xrightarrow{i?v} B'}{K \mid O \triangleright A \parallel B \xrightarrow{\mathsf{com}(i,v)} A' \parallel B'} \qquad \text{NCOMML}\ \frac{K \mid fId(B) \triangleright A \xrightarrow{i\uparrow j} A' \quad K \mid fId(A) \triangleright B \xrightarrow{i?j} B'}{K \mid O \triangleright A \parallel B \xrightarrow{\mathsf{ncom}} (\nu\, j)(A' \parallel B')}$$

$$\text{SCP2}\ \frac{K, j \mid O \triangleright A \xrightarrow{\mathsf{com}(i,v)} B}{K \mid O \triangleright (\nu\, j)A \xrightarrow{\mathsf{ncom}} (\nu\, j)B}\ j \in \{i, v\} \qquad \text{STR}\ \frac{A \equiv A' \quad B' \equiv B \quad K \mid O \triangleright A' \xrightarrow{\eta} B'}{K \mid O \triangleright A \xrightarrow{\eta} B}$$

Internal actor interaction is described via internal actions to permit monitors to differentiate these steps from the silent transitions. Transitions with $\mathsf{com}(i,v)$ labels are deduced via COMML (above) or the symmetric rule COMMR, whereas ncom-transitions are generated by the NCOMML, NCOMMR and SCP2 rules. Our semantics assumes standard structural equivalence as the ILTS equivalence relation, with axioms such as $A \equiv A \parallel \mathbf{0}$ and $A \parallel B \equiv B \parallel A$; transitions abstract over such states via rule STR. The remaining transitions are fairly standard.

A. Actor Structural Equivalence and Silent Actions

To show that our semantics is indeed an ILTS, we need to prove a few additional properties. Prop. VI.1 below shows that transitions abstract over structurally-equivalent states.

Proposition VI.1. *For any $A \equiv B$, whenever $K \mid O \triangleright A \xrightarrow{\eta} A'$ then there exists B' such that $K \mid O \triangleright B \xrightarrow{\eta} B'$ and $A' \equiv B'$.* ■

As a result of Prop. VI.2 below, we are guaranteed that any actor SUS instrumented via a mechanism that implements the semantics in Fig. 2 can safely abstract over (non-traceable) silent transitions because they are confluent w.r.t. other actions.

Proposition VI.2. *If $K \mid O \triangleright A \xrightarrow{\tau} A'$ and $K \mid O \triangleright A \xrightarrow{\eta} A''$, then either $\eta = \tau$ and $A' \equiv A''$ or there exists an actor system B and moves $K \mid O \triangleright A' \xrightarrow{\eta} B$ and $aft(K|O,\eta) \triangleright A'' \xrightarrow{\tau} B$.* ■

B. Deterministic and Non-deterministic Traceable Actions

Our ILTS interpretation treats input, output and internal communication as deterministic, justified by Prop. VI.3.

Proposition VI.3 (Determinacy)**.** *For all i, v, we have*

- *$K \mid O \triangleright A \xrightarrow{i!v} A'$ and $K \mid O \triangleright A \xrightarrow{i!v} A''$ implies $A' \equiv A''$*
- *$K \mid O \triangleright A \xrightarrow{i?v} A'$ and $K \mid O \triangleright A \xrightarrow{i?v} A''$ implies $A' \equiv A''$*
- *$K \mid O \triangleright A \xrightarrow{\mathsf{com}(i,v)} A'$ and $K \mid O \triangleright A \xrightarrow{\mathsf{com}(i,v)} A''$ implies $A' \equiv A''$* ■

In contrast, scope-extruding outputs and internal communication involving scoped names are *not* considered to be deterministic, *i.e.,* for all $i, j \in \text{PID}$, we have $\text{DET}(i\uparrow j) = \text{DET}(\mathsf{ncom}) = \mathsf{false}$. Exs. VI.1 and VI.2 illustrate why they are treated differently from other traceable actions.

Example VI.1. Consider the actor state $K \mid O \triangleright A_1$ where $j \in O$ and the running actor is defined as $A_1 \stackrel{\text{def}}{=} (\nu\, i)(\, i[\mathsf{rcv}\, x \to j!x.\mathbf{0}] \parallel i\langle v_1 \rangle \parallel i\langle v_2 \rangle\,)$ with $v_1 \neq v_2$; the actor identified by i is scoped by the outer construct $(\nu\, i)$. The actor at i can internally receive either value v_1 or v_1 via rules SCP2 and COMMR as follows:

$$K \mid O \triangleright A_1 \xrightarrow{\mathsf{ncom}} K \mid O \triangleright (\nu\, i)(i[\mathsf{rcv}\, x \to j!x.\mathbf{0} \triangleleft v_1] \parallel \mathbf{0} \parallel i\langle v_2 \rangle)$$
$$K \mid O \triangleright A_1 \xrightarrow{\mathsf{ncom}} K \mid O \triangleright (\nu\, i)(i[\mathsf{rcv}\, x \to j!x.\mathbf{0} \triangleleft v_2] \parallel i\langle v_1 \rangle \parallel \mathbf{0})$$

Since $v_1 \neq v_2$, the systems reached are *not* structurally equivalent: they exhibit a different observational behaviour by sending different payloads to the observer actor at j. ■

Example VI.2. Consider the actor system $K \mid O \triangleright A_2$ where $h \in O$ and the running actor is defined as $A_2 \stackrel{\text{def}}{=} (\nu\, i)\big(\, i[e_1] \parallel h\langle i \rangle\,\big) \parallel (\nu\, i)\big(\, i[e_2] \parallel h\langle i \rangle\,\big)$; name i is locally scoped twice and e_1 and e_2 exhibit different behaviour. The actor system $K \mid O \triangleright A_2$ can scope extrude name i by delivering the message $h\langle i \rangle$ in two possible ways using rules PARL, PARR and OPN as follows:

$$K \mid O \triangleright A_2 \xrightarrow{h\uparrow i} K \cup \{i\} \mid O \triangleright \big(\, i[e_1] \parallel \mathbf{0}\,\big) \parallel (\nu\, i)\big(\, i[e_2] \parallel h\langle i \rangle\,\big)$$
$$K \mid O \triangleright A_2 \xrightarrow{h\uparrow i} K \cup \{i\} \mid O \triangleright (\nu\, i)\big(\, i[e_1] \parallel h\langle i \rangle\,\big) \parallel (\, i[e_2] \parallel \mathbf{0})$$

Since the systems reached above are *not* structurally equivalent, they are possibly not behaviourally equivalent either. Particularly, once an observer learns of the new actor address i, it could interact with it by sending messages and subsequently observe different behaviour through the different e_1 and e_2. ■

Ex. VI.3 below showcases how the properties in Exs. I.1, I.3, I.4 and II.1 can be adapted to monitor for actor systems.

Example VI.3. With the values $\mathsf{req},\mathsf{ans},\mathsf{all},\mathsf{cls},\mathsf{init} \in \textsc{Atom}$, a server, expressed as actor i, can receive queries, $i?\mathsf{req}$, reply to an observer client located at j, $j!\mathsf{ans}$, and send messages to a resource manager, abstracted as an observer actor at address h, to either allocate more memory, $h!\mathsf{all}$, or close a connection, $h!\mathsf{cls}$. We can reformulate φ_4 from Ex. II.1 as

$$\varphi_6 \stackrel{\text{def}}{=} \max \mathsf{X}.\big([i?\mathsf{req}][j!\mathsf{ans}]\mathsf{X} \wedge ([h!\mathsf{cls}]\mathsf{ff} \vee [h!\mathsf{all}]\mathsf{ff})\big)$$

Assuming $\{i,j,h,k_1,k_2\} \subseteq K$ and $\{j,h\} \subseteq O$, consider the server implementation $K \mid O \triangleright A_{\text{srv}}$ that violates φ_6.

$$\begin{aligned} A_{\text{srv}} \stackrel{\text{def}}{=}\ & i[\mathsf{rcv}\,\mathsf{req} \to (k_1!\mathsf{init}.k_2!\mathsf{init}.j!\mathsf{ans})] \\ & \parallel k_1[\mathsf{rcv}\,\mathsf{init} \to h!\mathsf{all}] \parallel k_2[\mathsf{rcv}\,\mathsf{init} \to h!\mathsf{cls}.\mathbf{0}] \end{aligned}$$

This implementation can produce the history $\{t_1,t_2\}$ where we have $t_1{=}(i?\mathsf{req}).\mathsf{com}(k_1,\mathsf{init}).\mathsf{com}(k_2,\mathsf{init}).(j!\mathsf{ans}).(h!\mathsf{all})$ and $t_2{=}(i?\mathsf{req}).\mathsf{com}(k_1,\mathsf{init}).\mathsf{com}(k_2,\mathsf{init}).(j!\mathsf{ans}).(h!\mathsf{cls})$. Since, by Prop. VI.3, $\textsc{Det}(i?\mathsf{req}) = \textsc{Det}(i!\mathsf{ans}) = \mathsf{true}$, the visibility of the internal actions $\mathsf{com}(k_1,\mathsf{init})$ and $\mathsf{com}(k_2,\mathsf{init})$ suffices for the representative monitor $m_6 \stackrel{\text{def}}{=} (\!|\varphi_6|\!)$ to reject A_{srv}. This changes for $K'|O \triangleright (\nu\, k_1,k_2)(A_{\text{srv}})$ where $K' = K \setminus \{k_1,k_2\}$. The aforementioned traces would change to $t_3{=}(i?\mathsf{req}).\mathsf{ncom}.\mathsf{ncom}.(j!\mathsf{ans}).(h!\mathsf{all})$ and $t_4{=}(i?\mathsf{req}).\mathsf{ncom}.\mathsf{ncom}.(j!\mathsf{ans}).(h!\mathsf{cls})$. The obscured ncom events prohibit monitoring from determining whether behaviourally equivalent SUS states are reached after these transitions, thus soundly relate t_3 with t_4 in history $\{t_3,t_4\}$. ■

VII. Establishing Bounds

Despite the guarantees provided by Def. IV.3, Thms. V.3 and V.4 do not calculate the *number of monitored runs needed* to reject a violating system. This measure is crucial for an efficient implementation of the operational semantics of the monitor (Figs. 2 and 3) where history analysis is not invoked unnecessarily whenever a new trace is aggregated to the history. We therefore investigate whether there is a correlation between the syntactic structure of properties expressed in $\text{sHML}^{\vee}_{\text{DET}}$ and the number of partial traces required to conduct the verification. In particular, we study how this measure can be obtained through a *syntactic analysis* of the *disjunction operators* in the formula; for the purpose of tool construction, it is necessary for the analysis to be symbolic. Since we can only monitor for $\text{sHML}^{\vee}_{\text{DET}}$ formulae when the relevant internal actions are deterministic (see Ex. V.4), internal actions are elided in the subsequent discussion (they only make examples more cumbersome).

Example VII.1. Assume $\textsc{Det}(r) = \mathsf{true}$ and recall the specification $\varphi_2 \stackrel{\text{def}}{=} [r]\big([s]\mathsf{ff} \vee [a]\mathsf{ff}\big)$ from Ex. I.3 and its monitor $m_5 \stackrel{\text{def}}{=} r.\big(s.\mathsf{no} \oplus a.\mathsf{no}\big) = (\!|\varphi_2|\!)$ from Ex. V.2. Violating systems can produce the history $H = \{rs, ra\}$, which is enough for m_5 to reject. At the same time, no violating system for φ_2 can be rejected with fewer traces. Similarly, all violating systems for the formula $\varphi_5 \stackrel{\text{def}}{=} [r]\big([s]\mathsf{ff} \vee [a]\mathsf{ff}\big) \vee [a]\mathsf{ff}$ can be rejected via the 3-size history $\{rs, ra, a\}$ (modulo internal actions). ■

Although the evidence in Ex. VII.1 suggests that monitoring for a formula with n disjunctions requires $n{+}1$ executions to detect violations, this measure could be imprecise in general, due to a number of reasons. First, as a consequence of monitor passivity, there is no guarantee that the SUS will only produce the trace prefixes required to reject as it might also exhibit other behaviour. History bounds thus assume the *best case scenario* where *every* monitored run produces a *relevant* trace prefix. Second, not all SUS violations are justified by the same number of (relevant) trace prefixes. For instance, formulae such as $\varphi_1 \wedge \varphi_2$ are violated by systems that either violate φ_1 or φ_2 (but not necessarily both), and thus the number of relevant trace prefixes required to violate each subformula φ_i for $i \in 1..2$ might differ. This means that lower and upper bounds may not necessarily coincide.

Example VII.2. Consider $\varphi_7 \stackrel{\text{def}}{=} [r]\big([s]\mathsf{ff} \vee [a]\mathsf{ff}\big) \wedge [s]\mathsf{ff}$, a slight modification on φ_2. A representative monitor for φ_7 can reject violating systems that exhibit both trace prefixes ra and rs, but it can also reject others exhibiting the single prefix s via the subformula $[s]\mathsf{ff}$. This is problematic since our violating trace estimation needs to universally quantify over all systems, in order to adhere to a black-box treatment. ■

Recursive formulae complicate further the calculation of the executions required from the disjunctions present in a formula.

Example VII.3. φ_8 is a variation on φ_4, stating that *"if the system can allocate memory, then (i) it cannot also perform a close action and (ii) this property is invariant for all the states reached after servicing received queries."*

$$\begin{aligned} \varphi_8 \stackrel{\text{def}}{=}\ & \max \mathsf{X}.\big(\langle a\rangle\mathsf{tt} \Longrightarrow ([c]\mathsf{ff} \wedge [r][s]\mathsf{X})\big) \\ \equiv\ & \max \mathsf{X}.\big([a]\mathsf{ff} \vee ([c]\mathsf{ff} \wedge [r][s]\mathsf{X})\big) \end{aligned}$$

It contains one disjunction and $m_7 \stackrel{\text{def}}{=} \mathsf{rec}\mathsf{X}.\big(a.\mathsf{no} \oplus (r.s.\mathsf{X} \otimes c.\mathsf{no})\big) = (\!|\varphi_8|\!)$ can correctly monitor for it with no fewer than two trace prefixes. *E.g.* $p_1 \stackrel{\text{def}}{=} \mathsf{rec}\mathsf{X}.\big(r.s.\mathsf{X} + (a.\mathsf{X} + c.\mathbf{0})\big)$ from Ex. II.1 violates φ_8 and m_7 can detect this via the size-2 history $\{a,c\} \subseteq T_{p_1}$. But the same cannot be said for the violating system $p_{11} \stackrel{\text{def}}{=} a.\mathbf{0} + r.s.(a.\mathbf{0} + c.\mathbf{0})$. Since $p_{11} \stackrel{c}{\not\Longrightarrow}$, monitor m_7 cannot use the previous size-2 history and instead requires the size-3 history, $\{a, rsa, rsc\} \subseteq T_{p_{11}}$. Similarly, the violating system $p_{14} \stackrel{\text{def}}{=} a.\mathbf{0} + r.s.(a.\mathbf{0} + r.s.(a.\mathbf{0} + c.\mathbf{0}))$ can only be detected via a history containing the traces $\{a, rsa, rsrsa, rsrsc\}$. ■

Ex. VII.3 illustrates how, when universally quantifying over all systems, execution *upper* bounds cannot be easily determined from the structure of a formula. However, we show that the calculation of execution *lower* bounds from the formula structure is attainable. For instance, the lower bound

for a conjunction $\varphi_1 \wedge \varphi_2$ would be the least bound between the lower bounds of φ_1 and φ_2 respectively. Crucially, history lower bounds are invariant w.r.t. recursive formula unfolding.

Example VII.4. Recall φ_8 from Ex. VII.3 with a history lower bound of size 2, which is equal to the number of disjunctions in φ_8 plus 1 (as argued in Ex. VII.1). By the semantics in Fig. 1, the same systems also violate the unfolding of φ_8, *i.e.,*

$$\begin{aligned}\varphi_8' &\stackrel{\text{def}}{=} [a]\mathsf{ff} \vee \big([c]\mathsf{ff} \wedge [r][s](\max \mathsf{X}.([a]\mathsf{ff} \vee ([c]\mathsf{ff} \wedge [r][s]\mathsf{X})))\big)\\ &= [a]\mathsf{ff} \vee ([c]\mathsf{ff} \wedge [r][s]\varphi_8)\end{aligned}$$

since $\varphi_8 \equiv \varphi_8'$. A naive analysis would conclude that φ_8' contains 2 disjunctions, thereby requiring histories of size 3. But a compositional approach for such calculation, based on Ex. VII.2, would increase precision and allow us to conclude that lower bounds of size 2 suffice. Concretely, to reject a violating SUS for $\varphi_8' = [a]\mathsf{ff} \vee ([c]\mathsf{ff} \wedge [r][s]\varphi_8$, trace evidence is needed to determine violations for *both* sub-formulae $[a]\mathsf{ff}$ and $[c]\mathsf{ff} \wedge [r][s]\varphi_8$. Whereas 1 trace suffices to reject $[a]\mathsf{ff}$, determining the lower bounds for rejecting $[c]\mathsf{ff} \wedge [r][s]\varphi_8$ amounts to calculating the *least* lower bound required to reject either $[c]\mathsf{ff}$ or $[r][s]\varphi_8$. Since rejecting $[c]\mathsf{ff}$ requires only 1 trace, the total *lower* bound is that of $1+1=2$ traces, which is equal to that of φ_8. ■

The function $lb(-)$ formalises the calculation of history lower bounds based on the (compositional) syntactical analysis of formulae.

Definition VII.1. $lb(-) : \text{sHML}^{\vee}_{\text{DET}} \to \mathbb{N}$ is defined as follows:

$$\begin{array}{llll} lb(\mathsf{ff}) \stackrel{\text{def}}{=} 0 & lb(\max \mathsf{X}.\varphi) \stackrel{\text{def}}{=} lb(\varphi) & & lb([\alpha]\varphi) \stackrel{\text{def}}{=} lb(\varphi) \\ lb(\mathsf{tt}) \stackrel{\text{def}}{=} \infty & lb(\varphi \wedge \psi) \stackrel{\text{def}}{=} min\{lb(\varphi), lb(\psi)\} & & \\ lb(\mathsf{X}) \stackrel{\text{def}}{=} \infty & lb(\varphi \vee \psi) \stackrel{\text{def}}{=} lb(\varphi) + lb(\psi) + 1 & & \end{array}$$ ■

There is one further complication when calculating the number of trace prefixes required from the syntactic structure of formulae. Our implicit assumption has been that, for disjunctions $\varphi_1 \vee \varphi_2$, the incorrect system behaviour described by φ_1 and φ_2 is distinct. Whenever this is not the case, formulae do not observe the lower bound proposed above since φ_1 and φ_2 might be violated by common trace prefixes.

Example VII.5. Although analysing $\varphi_9 \stackrel{\text{def}}{=} [r]\mathsf{ff} \vee [r][s]\mathsf{ff}$ syntactically gives the lower bound 2, $m_8 \stackrel{\text{def}}{=} r.\mathsf{no} \oplus r.s.\mathsf{no} = (\!|\varphi_9|\!)$ rejects all violating systems with the single prefix rs. ■

We limit our calculations to a subset of RECHML ruling out overlapping violating behaviour across disjunctions. $\text{sHML}^{\vee}_{\text{NF}}$ (below) combines universal modalities and disjunctions into one construct, $\bigvee_{i\in I}[\alpha_i]\varphi_i$, to represent the formula $[\alpha_1]\varphi_1 \vee \cdots \vee [\alpha_n]\varphi_n$ for the finite set index $I=\{1,\ldots,n\}$.

Definition VII.2. $\text{sHML}^{\vee}_{\text{NF}} \subseteq$ RECHML is defined as:

$$\varphi, \psi \in \text{sHML}^{\vee}_{\text{NF}} ::= \mathsf{tt} \mid \mathsf{ff} \mid \varphi \wedge \psi \mid \bigvee_{i\in I}[\alpha_i]\varphi_i \mid \max \mathsf{X}.\varphi \mid \mathsf{X}$$

where $\forall i, j \in I$, we have $i \neq j$ implies $\alpha_i \neq \alpha_j$. ■

To faciliate the statement and establishment of results on history lowerbounds, we define an *explicit* witness-based violation relation $H \models_{\text{DET}} \varphi$ that avoids the existential quantifications over SUS histories of Defs. IV.1 and IV.2. The new judgement $H \models_{\text{DET}} \varphi$ corresponds to $p \notin [\![\varphi]\!]$ whenever $H \subseteq T_p$.

Definition VII.3. Given a predicate on TACT denoted as DET, the *violation relation*, denoted as $\models_{\text{DET}}$, is the least relation of the form $(\text{HST} \times \text{BOOL} \times \text{sHML}^{\vee}_{\text{DET}})$ satisfying the rules

$$\text{vF}\ \frac{H \neq \emptyset}{(H,\mathsf{f}) \models_{\text{DET}} \mathsf{ff}} \qquad \text{vMAX}\ \frac{(H,\mathsf{f}) \models_{\text{DET}} \varphi[\max \mathsf{X}.\varphi/\mathsf{X}]}{(H,\mathsf{f}) \models_{\text{DET}} \max \mathsf{X}.\varphi} \qquad \text{vANDL}\ \frac{(H,\mathsf{f}) \models_{\text{DET}} \varphi}{(H,\mathsf{f}) \models_{\text{DET}} \varphi \wedge \psi}$$

$$\text{vOR}\ \frac{(H,\mathsf{true}) \models_{\text{DET}} \varphi \quad (H,\mathsf{true}) \models_{\text{DET}} \psi}{(H,\mathsf{true}) \models_{\text{DET}} \varphi \vee \psi} \qquad \text{vANDR}\ \frac{(H,\mathsf{f}) \models_{\text{DET}} \psi}{(H,\mathsf{f}) \models_{\text{DET}} \varphi \wedge \psi}$$

$$\text{vUMPRE}\ \frac{H' = sub(H,\gamma) \quad \mathsf{f}' = \mathsf{f} \wedge \text{DET}(\alpha) \quad (H',\mathsf{f}') \models_{\text{DET}} [\alpha]\varphi}{(H,\mathsf{f}) \models_{\text{DET}} [\alpha]\varphi}$$

$$\text{vUM}\ \frac{H' = sub(H,\alpha) \quad \mathsf{f}' = \mathsf{f} \wedge \text{DET}(\alpha) \quad (H',\mathsf{f}') \models_{\text{DET}} \varphi}{(H,\mathsf{f}) \models_{\text{DET}} [\alpha]\varphi}$$

We read "H violates φ", $H \models_{\text{DET}} \varphi$, when $(H,\mathsf{true}) \models_{\text{DET}} \varphi$. ■

Thm. VII.1 shows that whenever a system p produces a history H that violates a formula φ, *i.e.,* $H \models_{\text{DET}} \varphi$, then p must also violate it, *i.e.,* $p \notin [\![\varphi]\!]$ (for *arbitrary* ILTSs). To show correspondence in the other direction, Thm. VII.2, we need to limit ILTSs to deterministic internal actions. The reason for this is, once again, the set of systems such as p_{10} from Ex. V.4 for which there is *no* history $H \subseteq T_{p_{10}}$ such that $H \models_{\text{DET}} \varphi_2$, even though $p_{10} \notin [\![\varphi_2]\!]$.

Theorem VII.1. *For all formulae* $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$, *if* $\big(\exists H \subseteq T_p$ *such that* $H \models_{\text{DET}} \varphi\big)$ *then* $p \notin [\![\varphi]\!]$. ■

Theorem VII.2. *Suppose* $\text{DET}(\gamma)=\mathsf{true}$ *for all* $\gamma \in$ IACT. *For all* $\varphi \in \text{sHML}^{\vee}_{\text{DET}}$, *if* $p \notin [\![\varphi]\!]$ *then* $\big(\exists H \subseteq T_p$ *s.t.* $H \models_{\text{DET}} \varphi\big)$. ■

The new judgment allows us to state and verify that disjunction sub-formulae must be violated by *disjoint* histories.

Proposition VII.3. *For all* $\varphi \vee \psi \in \text{sHML}^{\vee}_{\text{NF}}$, *if* $H \models_{\text{DET}} \varphi \vee \psi$ *then* $H = H' \uplus H''$ *such that* $H' \models_{\text{DET}} \varphi$ *and* $H'' \models_{\text{DET}} \psi$. ■

Thm. VII.4 establishes a lower bound on the trace prefixes required to detect violations for $\text{sHML}^{\vee}_{\text{NF}}$ formulae.

Theorem VII.4 (Lower Bounds)**.** *For all* $\varphi \in \text{sHML}^{\vee}_{\text{NF}}$ *and* $H \in$ HST, *if* $H \models_{\text{DET}} \varphi$ *then* $|H| \geq lb(\varphi) + 1$. ■

Example VII.6. Following Thm. VII.4, we can syntactically determine that $\varphi_2, \varphi_4, \varphi_8 \in \text{sHML}^{\vee}_{\text{NF}}$ cannot be violated (by any system) with fewer than 2 trace prefixes since $lb(\varphi_2) = lb(\varphi_4) = lb(\varphi_8) = 1$. ■

Thm. VII.4 also provides us with a simple syntactic check to determine whether $\text{sHML}^{\vee}_{\text{NF}}$ formulae are worth monitoring for, according to Def. V.1. Specifically, Cor. 1 shows that whenever $lb(\varphi) = \infty$, the formula φ is always satisfied, *i.e.,*

violations for it can *never* be detected, regardless of the system being runtime verified.

Corollary 1. *$lb(\varphi \in \text{sHML}^{\vee}_{\text{NF}})=\infty$ implies $\forall H \cdot H \not\models_{Det} \varphi$.* ■

Example VII.7. The formula $\varphi_\infty \stackrel{\text{def}}{=} \big(\max \mathsf{X}.[r][s]\mathsf{X}\big) \vee [a][c]\mathsf{ff}$ turns out to be a tautology and, accordingly, $lb(\varphi_\infty)=\infty$. ■

Finally, we note that although a minimum of n trace prefixes might be required by Def. VII.1 for analysis, the SUS might need to be executed *more* than n times to obtain these prefixes. Intuitively, this is caused by redundancies in the monitors (caused by the compositional method of monitor synthesis) and the incremental manner in which monitor instrumentation record trace prefixes, as illustrated in Ex. VII.8.

Example VII.8. Assuming $\text{DET}(a)=\text{true}$, consider φ_{10}, describing the property *"after any number of serviced queries interspersed by sequences of memory allocations, a system that can allocate memory cannot also perform a close action."*

$$\varphi_{10} \stackrel{\text{def}}{=} \max \mathsf{X}.\big([r][s]\mathsf{X} \wedge [a]\mathsf{X} \wedge ([a]\mathsf{ff} \vee [c]\mathsf{ff})\big)$$

When synthesising φ_{10}, we get the monitor $(\!|\varphi_{10}|\!) = m_2 \stackrel{\text{def}}{=} \text{rec}\mathsf{X}.\big(r.s.\mathsf{X} \otimes a.X \otimes (a.\mathsf{no} \oplus c.\mathsf{no})\big)$ from Ex. III.2. The system $p_{13} \stackrel{\text{def}}{=} \text{rec}\mathsf{X}.r.s.\mathsf{X}+a.X+a.c.\mathbf{0}$ violates φ_{10}, and m_2 can reject it via the history $H=\{rsaa, rsac\} \subseteq T_{p_{13}}$, in line with Thm. VII.4 since $lb(\varphi_{10})+1=2$ trace prefixes. However, the incremental manner with which traces are aggregated (Sec. III) requires that, whenever $rsaa \in H$, then $rsa \in H$ as well. This is due to the fact that for the trace $rsa\cdots$, we always have $\emptyset \triangleright (\varepsilon, m_2) \triangleleft p_{14} \stackrel{rsa}{\Longrightarrow} \emptyset \triangleright (rsa, \mathsf{no}) \triangleleft p'_{14}$ during the first monitored execution. Thus, although 2 prefixes are sufficient to detect a violation, the operational mechanism for aggregating the traces for analysis forces us to observe at least *3* SUS executions to gather the necessary traces for analysis. ■

VIII. Related Work

Various bodies of work employ monitors over multiple runs for RV purposes. The most prominent target *Hyperproperties*, *i.e.,* properties describing sets of traces called *hypertraces*, used to describe safety and privacy requirements [56]. Finkbeiner *et al.* [57] investigate the monitorability of hyperproperties expressed in HyperLTL [58] and identify three classes for monitoring hypertraces: the bounded sequential, the unbounded sequential and the parallel classes. They also develop a monitoring tool [59] that analyses hypertraces sequentially by converting an alternation-free HyperLTL formula into an alternating automaton that is executed over permutations of the observed traces. They show that deciding monitorability for alternation-free HyperLTL formulae in this class is PSPACE-complete but undecidable in general. Our setup fits their unbounded sequential class because monitors receive each trace in sequence, and a SUS may exhibit an unbounded number of traces. Agrawal *et al.* [60] give a semantic characterisation for monitorable HyperLTL hyperproperties called k-safety. They also identify syntactic HyperLTL fragments and show they are k-safety properties, backed up by a monitor synthesis algorithm that generates a combination of petri-nets and LTL_3 monitors [61]. Stucki *et al.* [62] show that many properties in HyperLTL involving quantifier alternation cannot be monitored for. They also present a methodology for properties with one alternation by combining static verification and RV: the static part extracts information about the set of traces that the SUS can produce (*i.e.,* branching information about the number of traces in the SUS, expressed as a symbolic execution tree) that is used by monitors to convert quantifications into k-(trace)-quantifications. More recently, in [44] Aceto *et al.* study the adaptation of the linear-time RECHML [11] to hypertraces. Their definition of monitorability follows the same template to that of defs. IV.1, IV.2 and V.1. They study a syntax-directed monitor synthesis similar to ours, exploring both centralised and decentralised alternatives where choreographed monitors interact with one another.

Despite the similarities of using multi-run monitoring, these works differ from ours in a number of ways. For instance, the methods used are very different. Our monitor synthesis algorithm is directly based on the formula syntax and does not rely on auxiliary models such as alternating automata or petri-nets, which facilitates syntactic-based proofs. The results presented are also substantially different. Although [60], [62] prove that their monitor synthesis algorithm is sound, neither work considers completeness results, maximality or execution lower bound estimation. More importantly, our target logic, RECHML, is intrinsically different from (linear-time) hyperlogics since it (and other branching-time logics) is interpreted over LTSs, whereas hyperlogics are defined over sets of traces. which inherently coarser than an LTSs. For instance, the systems $a.b.\mathbf{0}+a.c.\mathbf{0}$ and $a.(b.\mathbf{0}+c.\mathbf{0})$ are described by different LTSs but have an identical trace-based model, *i.e.,* $\{ab, ac\}$; this was a major source of complication for our technical development. Even for deterministic LTSs where the system $a.b.\mathbf{0}+a.c.\mathbf{0}$ is disallowed, it remains unclear how the two types of logics correspond. For one, hyperlogics employ existential and universal quantifications over traces, which are absent from our logic. If we had to normalise these differences (*i.e.,* rule out trace quantifications), a reasonable mapping would be to take a a linear-time interpretation, $[\![\varphi]\!]_{\text{LT}}$ [11], [23] for every RECHML branching-time property φ, and require it to hold for all of its traces: For all φ and deterministic systems p, we would then expect $p \in [\![\varphi]\!]$ iff $T_p \in [\![\varphi]\!]_{\text{LT}}$. But even this correspondence fails. For instance, the branching-time $[a]\mathsf{ff} \vee [b]\mathsf{ff}$, describes *systems* that cannot perform both a and b actions and $a.\mathbf{0}+b.\mathbf{0}$ clearly violates it. However, with a linear-time interpretation, this formula denotes a *tautology*: it is satisfied by *all* traces since they are necessarily either not prefixed with an a action or with a b action. There are, however, notable similarities between our history evaluation (Fig. 3) and team semantics for temporal logics [63], [64], and this relationship is worth further investigation.

The closest work to ours is [25], where Aceto *et al.* give a framework to extend the capabilities of monitors. They study monitorability under a grey-box assumption where, at runtime, a monitor has access to additional SUS information, linked

to the system's states, in the form of decorated states. The additional state information is parameterised by a class of *conditions* that represent different situations, such as access to information about that state gathered from previous system executions. Other works have also examined how to use prior knowledge about the SUS to extend monitorability in the linear-time and branching-time settings, *e.g.* [24], [65]. In contrast, we treat the SUS as a black-box.

Multiple traces are also used to runtime verify traces with imprecise event ordering [66], [67], [68], [69] due to interleaved executions of components. Parametric trace slicing [67], [68] infers additional traces from a trace with interleaved events by traversing the original trace and dispatching events to the corresponding slice. Attard *et al.* [69] partition the observed trace at the instrumentation level by synthesising monitors attached to specific system components; they hint at how this could enhance the monitoring expressive power for certain properties but do not prove any monitorability results. Despite their relevance, all traces in [66], [67], [68], [69] are extracted from a *single* execution.

In [70], Abramsky studies testing on multiple, yet finite, copies of the same system, combining the information from multiple runs. Our approach differs in three key aspects. Firstly, our multiple executions correspond to creating multiple copies of the system from its initial state; Abramsky allows copies to be created at *any point* of the execution. Secondly, tests are composed using parallel composition, can steer the execution of the SUS and can detect refusals. In contrast, our monitors are composed using an instrumentation relation: they are passive and their verdicts are evidence-based (*i.e.,* what happened, not what could *not* have happened). Third, the visibility afforded by monitor instrumentation considered in this work is larger than that obtained via parallel composition. Consult [71] and [20, Sec. 9.1] for a detailed comparison between tests and monitors.

Akin to our history analysis in Sec. III-B, Silva *et al.* [5] investigate combining traces produced by the same system over a number of runs to create temporal models that approximates the SUS's behaviour which can then be used to model check for branching-time properties. Their approach is *not* sound as the generated model may violate properties that are not violated by the actual system. The authors advise using their approach as a complement to software testing to suggest possible problems.

IX. Conclusion

We propose a framework to systematically extend RV to verify branching-time properties. This is in sharp contrast to most research on RV, which centers around monitoring linear-time properties [21], [22]. As shown in [11], the class of monitorable linear-time (regular) properties is syntactically larger than that of monitorable branching-time properties, explaining, in part, why the linear-time setting appears less restrictive when runtime verified. For instance, linear-time properties that are monitorable for violations are closed under disjunctions, $\varphi \vee \phi$, and existential modalities, $\langle \alpha \rangle \varphi$, as these can be encoded in an effective, if not efficient, manner [11], [72], albeit in a setting with finite sets of actions. In contrast, disjunctions and existential modalities in a branching-time setting cannot be encoded in terms of other recHML constructs.

Our work shows that the limitations of runtime verifying branching-time properties can be mitigated by observing multiple system executions. Our results demonstrate that monitors can extract sufficient information over multiple runs to correctly detect the violation of a class of branching-time properties that may contain disjunctions (Thm. V.3). We also prove that the monitorable fragment $\mathrm{sHML}^{\vee}_{\mathrm{DET}}$ (Def. V.2) is maximally expressive. In particular, every property that can be monitored correctly using our monitoring framework can always be expressed as a formula in $\mathrm{sHML}^{\vee}_{\mathrm{DET}}$. Such a syntactic characterisation of monitorable properties is useful for tool construction. It is worth pointing out that an implementation based on our theoretical framework could relax the assumptions used only to attain completeness and maximality results; *e.g.* instead of assuming that all internal actions are deterministic, a tool could adopt a pragmatic stance and simply stop monitoring as soon as a non-deterministic internal action is encountered, which would still yield a sound (but incomplete) monitor. To validate the realisability of our multi-run monitoring RV framework over the ILTS model of Sec. II, we outline a possible instantiation to actor-based systems in Sec. VI. We also show that the number of expected runs required to effect the runtime analysis can be calculated from the structure of the formula being verified (as opposed to other means [62]); see Thm. VII.4. We are unaware of similar results in the RV literature.

Future Work: We plan to investigate how our results can be extended by considering more of a grey-box view of the system, in order to combine our machinery with techniques from existing work, such as that of Aceto *et al.* [25]. We will also study strategies to optimise the collection of relevant SUS traces. Depending on the application, one might seek to either maximize the information collected from every execution (*e.g.* by continuing to monitor the same execution after a trace prefix is added to the history) or minimize the runtime during which the monitor is active. This investigation will be used for tool construction, possibly by extending existing (single-run) open-source monitoring tools for recHML such as detectEr [12], [47] that already target actor systems. We also plan to extend our techniques to other graph-based formalisms such as Attack/Fault Trees [73], [74], [75], [76] used in cybersecurity, which often necessitate verification at runtime.

References

[1] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT press, 1999.

[2] C. Baier, J.-P. Katoen, and K. G. Larsen, *Principles of model checking*. MIT press, 2008.

[3] Y. Kesten and A. Pnueli, "A compositional approach to ctl* verification," *TCS*, vol. 331, no. 2-3, pp. 397–428, 2005.

[4] A. Pnueli and A. Zaks, "Psl model checking and run-time verification via testers," in *FM 2006: Formal Methods*, J. Misra, T. Nipkow, and E. Sekerinski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 573–586.

[5] P. S. da Silva and A. C. de Melo, "Model checking merged program traces," *Electronic Notes in Theoretical Computer Science*, vol. 240, pp. 97–112, 2009, SBMF.

[6] T. L. Hinrichs, A. P. Sistla, and L. D. Zuck, "Model check what you can, runtime verify the rest," in *HOWARD-60*, ser. EPiC Series in Computing. EasyChair, 2014, vol. 42, pp. 234–244.

[7] W. Ahrendt, J. M. Chimento, G. J. Pace, and G. Schneider, "A specification language for static and runtime verification of data and control properties," in *FM*, ser. LNCS, vol. 9109. Springer, 2015, pp. 108–125.

[8] A. Francalanza, L. Aceto, and A. Ingólfsdóttir, "Monitorability for the Hennessy-Milner logic with recursion," *FMSD*, vol. 51, no. 1, pp. 87–116, 2017.

[9] A. Desai, T. Dreossi, and S. A. Seshia, "Combining model checking and runtime verification for safe robotics," in *RV*, ser. LNCS, vol. 10548. Springer, 2017, pp. 172–189.

[10] K. Kejstová, P. Rockai, and J. Barnat, "From model checking to runtime verification and back," in *RV*, ser. LNCS, vol. 10548. Springer, 2017, pp. 225–240.

[11] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen, "Adventures in Monitorability: From Branching to Linear Time and Back Again," *PACMPL*, vol. 3, no. POPL, pp. 52:1–52:29, 2019.

[12] D. P. Attard, L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen, "Better Late Than Never or: Verifying Asynchronous Components at Runtime," in *IFIP*, ser. LNCS, vol. 12719. Springer, 2021, pp. 207–225.

[13] S. Stucki, C. Sánchez, G. Schneider, and B. Bonakdarpour, "Gray-box monitoring of hyperproperties with an application to privacy," *Formal Methods Syst. Des.*, vol. 58, no. 1-2, pp. 126–159, 2021.

[14] G. Audrito, F. Damiani, V. Stolz, G. Torta, and M. Viroli, "Distributed runtime verification by past-ctl and the field calculus," *Journal of Systems and Software*, vol. 187, p. 111251, 2022.

[15] A. Ferrando and V. Malvone, "Towards the combination of model checking and runtime verification on multi-agent systems," in *PAAMS*, ser. LNCS, vol. 13616. Springer, 2022, pp. 140–152.

[16] L. Aceto, I. Cassar, A. Francalanza, and A. Ingólfsdóttir, "Bidirectional runtime enforcement of first-order branching-time properties," *Log. Methods Comput. Sci.*, vol. 19, no. 1, 2023.

[17] F. B. Schneider, "Enforceable Security Policies," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 1, 2000.

[18] J. Ligatti, L. Bauer, and D. Walker, "Edit automata: enforcement mechanisms for run-time security policies," *IJIS*, vol. 4, no. 1-2, 2005.

[19] N. Bielova and F. Massacci, "Do you really mean what you actually enforced? edited automata revisited," *IJIS*, vol. 10, no. 4, p. 239–254, 2011.

[20] A. Francalanza, "A Theory of Monitors," *Inf. Comput.*, vol. 281, p. 104704, 2021.

[21] E. Bartocci, Y. Falcone, A. Francalanza, and G. Reger, "Introduction to Runtime Verification," in *Lectures on Runtime Verification - Introductory and Advanced Topics*, ser. LNCS. Springer, 2018, vol. 10457, pp. 1–33.

[22] M. Leucker and C. Schallhart, "A brief account of runtime verification," *JLAMP*, vol. 78, no. 5, pp. 293–303, 2009.

[23] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen, "An operational guide to monitorability with applications to regular properties," *Softw. Syst. Model.*, vol. 20, no. 2, pp. 335–361, 2021.

[24] ——, "The Best a Monitor Can Do," in *CSL*, ser. LIPIcs, vol. 183. Schloss Dagstuhl, 2021, pp. 7:1–7:23.

[25] L. Aceto, A. Achilleos, A. Francalanza, and A. Ingólfsdóttir, "A Framework for Parameterized Monitorability," in *FOSSACS*, ser. LNCS, vol. 10803. Springer, 2018, pp. 203–220.

[26] X. Zhang, M. Leucker, and W. Dong, "Runtime verification with predictive semantics," in *NASA Formal Methods*, ser. LNCS, vol. 7226, 2012, pp. 418–432.

[27] P. Selinger, "First-order axioms for asynchrony," in *CONCUR*, ser. LNCS, 1997, vol. 1243, pp. 376–390.

[28] K. Honda and M. Tokoro, "An object calculus for asynchronous communication," in *ECOOP*, vol. 512, 2006, pp. 133–147.

[29] D. Kozen, "Results on the Propositional mu-Calculus," *TCS*, vol. 27, pp. 333–354, 1983.

[30] K. G. Larsen, "Proof systems for satisfiability in hennessy-milner logic with recursion," *TCS*, vol. 72, no. 2, pp. 265 – 288, 1990.

[31] S. Cranen, J. F. Groote, J. J. A. Keiren, F. P. M. Stappers, E. P. de Vink, W. Wesselink, and T. A. C. Willemse, "An Overview of the mCRL2 Toolset and Its Recent Advances," in *TACAS*, ser. LNCS, vol. 7795. Springer, 2013, pp. 199–213.

[32] G. Behrmann, A. David, and K. G. Larsen, *A Tutorial on Uppaal*. Springer, 2004, pp. 200–236.

[33] L. Aceto, A. Achilleos, D. P. Attard, L. Exibard, A. Francalanza, and A. Ingólfsdóttir, "A monitoring tool for linear-time μhml," *Sci. Comput. Program.*, vol. 232, p. 103031, 2024.

[34] A. Achilleos, A. Francalanza, and J. Xuereb, "If At First You Don't Succeed: Extended Monitorability through Multiple Executions," 2025. [Online]. Available: https://arxiv.org/abs/2306.05229v3

[35] N. Yoshida, K. Honda, and M. Berger, "Linearity and bisimulation," *JLAMP*, vol. 72, no. 2, pp. 207–238, 2007.

[36] M. Hennessy, *A distributed Pi-calculus*. Cambridge University Press, 2007.

[37] B. Alpern and F. B. Schneider, "Recognizing Safety and Liveness," *Distributed Comput.*, vol. 2, no. 3, pp. 117–126, 1987.

[38] A. Francalanza, "Consistently-Detecting Monitors," in *CONCUR*, ser. LIPIcs, vol. 85. Schloss Dagstuhl, 2017, pp. 8:1–8:19.

[39] A. Francalanza, L. Aceto, A. Achilleos, D. P. Attard, I. Cassar, D. D. Monica, and A. Ingólfsdóttir, "A foundation for runtime monitoring," in *RV*, ser. LNCS, vol. 10548. Springer, 2017, pp. 8–29.

[40] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and S. Ö. Kjartansson, "Determinizing monitors for HML with recursion," *JLAMP*, vol. 111, p. 100515, 2020.

[41] Y. Falcone, J. Fernandez, and L. Mounier, "What can you verify and enforce at runtime?" *Int. J. Softw. Tools Technol. Transf.*, vol. 14, no. 3, pp. 349–382, 2012.

[42] T. A. Henzinger and N. E. Saraç, "Quantitative and approximate monitoring," in *LICS*. IEEE, 2021, pp. 1–14.

[43] A. Castañeda and G. V. Rodríguez, "Asynchronous wait-free runtime verification and enforcement of linearizability," in *PODC*. ACM, 2023, pp. 90–101.

[44] L. Aceto, A. Achilleos, E. Anastasiadi, A. Francalanza, D. Gorla, and J. Wagemaker, "Centralized vs Decentralized Monitors for Hyperproperties," in *CONCUR*, ser. LIPIcs, vol. 311, 2024, pp. 4:1–4:19.

[45] A. Ferrando and R. C. Cardoso, "Towards partial monitoring: Never too early to give in," *Science of Computer Programming*, vol. 240, p. 103220, 2025.

[46] M. Amara, G. Bernardi, M. Foughali, and A. Francalanza, "A Theory of (Linear-Time) Timed Monitors," in *39th European Conference on Object-Oriented Programming (ECOOP 2025)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2025, (to appear).

[47] L. Aceto, A. Achilleos, D. P. Attard, L. Exibard, A. Francalanza, and A. Ingólfsdóttir, "A Monitoring Tool for Linear-Time μHML," in *COORDINATION*, ser. LNCS, vol. 13271. Springer, 2022, pp. 200–219.

[48] C. Hewitt, P. B. Bishop, and R. Steiger, "A universal modular ACTOR formalism for artificial intelligence," in *IJCAI*, 1973, pp. 235–245.

[49] G. A. Agha, *ACTORS - A Model of Concurrent Computation in Distributed Systems*. MIT Press, 1990.

[50] F. Cesarini and S. Thompson, *Erlang Programming - A Concurrent Approach to Software Development*. O'Reilly, 2009.

[51] J. Goodwin, *Learning Akka: Build Fault-tolerant, Concurrent, and Distributed Applications with Akka*, ser. Community experience distilled. Packt Publishing, 2015.

[52] S. Juric, *Elixir in Action, Third Edition*. Manning, 2024.

[53] Apple Inc. and the Swift project authors, *The Swift Programming Language (6.0 beta)*, 2024.

[54] D. Sangiorgi and D. Walker, *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001.

[55] J. Bengtson and J. Parrow, "Formalising the pi-calculus using nominal logic," *Log. Methods Comput. Sci.*, vol. 5, no. 2, 2009.

[56] M. R. Clarkson and F. B. Schneider, "Hyperproperties," *JCS*, vol. 18, no. 6, pp. 1157–1210, 2010.

[57] B. Finkbeiner, C. Hahn, M. Stenger, and L. Tentrup, "Monitoring hyperproperties," *FMSD*, vol. 54, no. 3, pp. 336–363, 2019.

[58] M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez, "Temporal logics for hyperproperties," in *POST*, ser. LNCS, vol. 8414. Springer, 2014, pp. 265–284.

[59] B. Finkbeiner, C. Hahn, M. Stenger, and L. Tentrup, "Rvhyper: A runtime verification tool for temporal hyperproperties," in *TACAS (2)*, ser. LNCS, vol. 10806. Springer, 2018, pp. 194–200.

[60] S. Agrawal and B. Bonakdarpour, "Runtime Verification of k-Safety Hyperproperties in HyperLTL," in *IEEE*, 2016, pp. 239–252.

[61] A. Bauer, M. Leucker, and C. Schallhart, "Runtime verification for LTL and TLTL," *ACM*, vol. 20, no. 4, pp. 14:1–14:64, 2011.

[62] S. Stucki, C. Sánchez, G. Schneider, and B. Bonakdarpour, "Gray-box monitoring of hyperproperties with an application to privacy," *FMSD*, pp. 1–34, 2021.
[63] A. Krebs, A. Meier, J. Virtema, and M. Zimmermann, "Team Semantics for the Specification and Verification of Hyperproperties," in *MFCS*, ser. LIPIcs, vol. 117. Schloss Dagstuhl, 2018, pp. 10:1–10:16.
[64] J. Virtema, J. Hofmann, B. Finkbeiner, J. Kontinen, and F. Yang, "Linear-Time Temporal Logic with Team Semantics: Expressivity and Complexity," in *IARCS*, ser. LIPIcs, vol. 213. Schloss Dagstuhl, 2021, pp. 52:1–52:17.
[65] T. A. Henzinger and N. E. Saraç, "Monitorability Under Assumptions," in *RV*, ser. LNCS, vol. 12399. Springer, 2020, pp. 3–18.
[66] S. Wang, A. Ayoub, O. Sokolsky, and I. Lee, "Runtime Verification of Traces under Recording Uncertainty," in *RV*, ser. LNCS, vol. 7186. Springer, 2011, pp. 442–456.
[67] F. Chen and G. Rosu, "Parametric Trace Slicing and Monitoring," in *TACAS*, ser. LNCS, vol. 5505. Springer, 2009, pp. 246–261.
[68] H. Barringer, Y. Falcone, K. Havelund, G. Reger, and D. E. Rydeheard, "Quantified Event Automata: Towards Expressive and Efficient Runtime Monitors," in *FM*, ser. LNCS, vol. 7436. Springer, 2012, pp. 68–84.
[69] D. P. Attard and A. Francalanza, "Trace Partitioning and Local Monitoring for Asynchronous Components," in *SEFM*, ser. LNCS, vol. 10469. Springer, 2017, pp. 219–235.
[70] S. Abramsky, "Observation equivalence as a testing equivalence," *TCS*, vol. 53, no. 2, pp. 225–241, 1987.
[71] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen, "Testing equivalence vs. runtime monitoring," in *Models, Languages, and Tools for Concurrent and Distributed Programming*, ser. Lecture Notes in Computer Science, vol. 11665. Springer, 2019, pp. 28–44.
[72] ——, "The Cost of Monitoring Alone," in *From Reactive Systems to Cyber-Physical Systems*, ser. LNCS, vol. 11500, 2019, pp. 259–275.
[73] B. Schneier, "Attack Trees," *Dr. Dobb's Journal*, 1999.
[74] E. Ruijters and M. Stoelinga, "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools," *Comput. Sci. Rev.*, vol. 15, pp. 29–62, 2015.
[75] M. Audinot, S. Pinchinat, and B. Kordy, "Is My Attack Tree Correct?" in *ESORICS*, ser. LNCS, vol. 10492, 2017, pp. 83–102.
[76] F. Kammüller, "Attack Trees in Isabelle," in *ICICS*, ser. LNCS, vol. 11149, 2018, pp. 611–628.