Centralized vs Decentralized Monitors for Hyperproperties

Luca Aceto
Dept. of Computer Science, Reykjavik University, Iceland
Gran Sasso Science Institute, L'Aquila, Italy

Antonis Achilleos
Dept. of Computer Science, Reykjavik University, Iceland

Elli Anastasiadi
Uppsala University, Sweden

Adrian Francalanza
University of Malta, Malta

Daniele Gorla
Dept. of Computer Science, "Sapienza" University of Rome, Italy

Jana Wagemaker
Dept. of Computer Science, Reykjavik University, Iceland

Abstract

This paper focuses on the runtime verification of hyperproperties expressed in Hyper-recHML, an expressive yet simple logic for describing properties of sets of traces. To this end, we consider a simple language of monitors that observe sets of system executions and report verdicts w.r.t. a given Hyper-recHML formula. We first employ a unique omniscient monitor that centrally observes all system traces. Since centralised monitors are not ideal for distributed settings, we also provide a language for decentralized monitors, where each trace has a dedicated monitor; these monitors yield a unique verdict by communicating their observations to one another. For both the centralized and the decentralized settings, we provide a synthesis procedure that, given a formula, yields a monitor that is correct (i.e., sound and violation complete). A key step in proving the correctness of the synthesis for decentralized monitors is a result showing that, for each formula, the synthesized centralized monitor and its corresponding decentralized one are weakly bisimilar for a suitable notion of weak bisimulation.

2012 ACM Subject Classification Theory of computation → Operational semantics; Theory of computation → Modal and temporal logics; Theory of computation → Logic and verification

Keywords and phrases Runtime Verification, hyperlogics, decentralization

Digital Object Identifier 10.4230/LIPIcs...

Funding This work has been supported by the project 'Mode(l)s of Verification and Monitorability' (MoVeMent) (grant No 217987) of the Icelandic Research Fund. Elli Anastasiadi's research has been supported by grant VR 2020-04430 of the Swedish Research Council.

1 Introduction

Runtime verification (RV) [12] is a verification technique that observes system executions to determine whether some given specification is satisfied or violated. This runtime analysis is usually conducted by a computational entity called a *monitor* [33]. RV is a lightweight verification technique that is carried out as the system under observation executes, thereby avoiding scalability issues caused by the state-explosion problem, as is the case for model checking. Recently, RV has been extended to parallel set-ups [17, 24, 45], and a large body of work in that setting aims to verify *hyperproperties* at runtime [1, 18, 19, 27, 30].

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Hyperproperties [27] are sets of *hypertraces*, *i.e.* sets of traces that may be seen as describing different system executions or the contributions of different sequential processes to a system execution. As argued in [22], many properties of concurrent and distributed systems can be viewed as hyperproperties. When verifying hyperproperties at runtime, several traces (i.e. several execution sequences) can be observed instead of just one, possibly at the same time. Several extensions of temporal logics, such as HyperLTL, HyperCTL* [26], Hyper^{2}LTL [14], have been defined to express hyperproperties. Extensions of standard logics to hyper properties also include variations of the μ-calculus, such as [1], setting the basis for the logic used in this paper, and [36], which studies an asynchronous semantics.

Since they were proposed by Clarkson and Schneider in [27], hyperproperties have become a fundamental, trace-based formalism for expressing security and privacy properties, verified using static and dynamic techniques [10, 14, 15, 18, 22, 23, 25, 30] implemented in a variety of tools [13,15,29]. There is a large body of work, such as [10,23,37], detailing several algorithms for monitoring (fragments of) hyperlogics under different assumptions and providing several correctness guarantees. However, these proposals either construct a centralized monitoring algorithm that has access to all traces in the observed hypertrace, or verify single trace properties, over a distributed set-up[1]. Having an omniscient monitor simplifies the runtime analysis since the monitoring algorithm can compare all traces as needed by simply accessing different parts of its local memory. But this power comes with drawbacks. For starters, centralized monitors are unrealistic for distributed systems, where trace analysis is typically localised to network nodes so as to minimize communication across locations. Moreover, centralized monitors create single points of failure during verification [8]. Furthermore, it can be problematic to store all the traces locally, especially in light of the wide availability of multi-core systems. The goal of the decentralized monitor synthesis from logical specifications presented in this paper is to permit distributed monitor choreographies with *local* trace views whose components communicate in order to verify *global* properties (such as hyperproperties). Decentralized monitors have been shown to avoid high contentions leading to vastly improved scalability [8]. They also offer better privacy guarantees whenever they are stationed locally at the nodes where the respective traces are generated [35, 39]. To the best of our knowledge, such a message-passing monitoring set-up has never been studied for the purpose of verifying hyperproperties so far.

In this paper, we study procedures for the *automated synthesis of centralized and decentralized monitors* from hyperproperties described in the logic Hyper-recHML [1]. This logic extends the linear-time [51] μ-calculus [40] (also known as Hennessy-Milner logic with recursion [44]) with constructs to describe properties of hypertraces inspired by the work on HyperLTL (namely variables ranging over traces, modal operators parametrized by trace variables, matching/mismatching between trace variables, and existential and universal quantification over them). Hyper-recHML can describe hyperproperties not expressible in HyperLTL or HyperCTL*, such as properties that speak about consensus (see Example 2.2) and periodicity (see Example 2.3). Furthermore, Hyper-recHML supports a general, syntax-driven monitor synthesis that can handle both the aforementioned hyperproperties, at least in the centralized case (see also the discussion in Section 5).

In both the centralized and decentralized set-ups, we work in the parallel model [30], where a fixed number of system executions is processed in parallel by monitors in an online fashion. We specify monitors using a process-algebraic formalism that builds on the one presented in [5, 34] to define a class of monitors called regular. Such monitors are easy to describe,

[1] See e.g. [20, 21, 31, 35] for distributed monitoring algorithms for classic trace-based logics.

resemble (alternating) automata, and have sufficient expressive power to provide standard monitoring guarantees. Moreover, their algebraic structure supports the compositional definition of their operational semantics and monitor synthesis procedures from formulas, building on previous work relating algebraic process calculi with RV [6,9,16,32,33,38,42,43].

In the centralized case, for each formula in the fragment of Hyper-recHML limited to greatest-fixed-point operators, our synthesis procedure yields a monolithic monitor that has access to all the traces in an observed hypertrace. However, in order to synthesize decentralized monitors for a sufficiently expressive fragment of the logic, it is necessary to extend the monitor capabilities with communication, as shown already in [1]. For instance, to monitor for the property "If there is a trace where event a occurs, then there exists another trace where event b does not occur thereafter", monitors observing different traces need to communicate to record that event a occurred in some trace at some point and that there is some trace where b does not occur from that point onwards. Allowing monitors to send and receive messages significantly complicates their operational semantics (see Section 4), the monitor synthesis procedure (see Section 4.2), and all consequent proofs. The operational semantics for communicating monitors is one of the main contributions of the paper since its design is crucial to obtain the correctness guarantees provided by the synthesis procedure for decentralized monitors. In particular, the semantics of decentralized monitors and their synthesis from formulas have to be designed carefully to ensure that monitors are reactive (they are always ready to process any system event) and input-enabled (they can always receive any input from other monitors in their environment), properties that are desirable in any decentralized RV set-up.

We show that both *the centralized and the decentralized monitor synthesis procedures are correct*. More precisely, the monitors synthesized from formulas are *sound* and *violation-complete*, meaning that (1) if the monitor synthesized from a formula φ reports a positive (resp., negative) verdict when observing a hypertrace T, then T does (resp., does not) satisfy φ, and (2) if T does not satisfy φ, then its associated monitor will report a negative verdict when observing T (see Theorems 3.2 and 3.3, and Corollaries 4.2 and 4.3). The proof of correctness in the decentralized case is considerably more technical than the corresponding proof in the centralized setting, due to the intricate communication semantics. To address the resulting technical challenges, we develop a proof strategy where we prove the correctness of the decentralized monitor synthesis procedure using the centralized one as a yardstick.

This methodology is one of the key contributions we offer in this study. More precisely, in Section 4.1 *we identify six properties of a decentralized monitor synthesis that make it 'principled'* (see Definition 4.5) and we show that, when a decentralized monitor synthesis is principled, the centralized and decentralized monitors synthesized from a formula are related by a suitable notion of weak bisimulation (Theorem 4.6). Apart from supporting the definition of decentralized monitor synthesis procedures, this result allows us to reduce the correctness of our decentralized monitor synthesis to that of the centralized one, which can in turn drive the definition of further synthesis procedures in future work. We also conjecture that our methodology provides a path to proving similar results for other models of communicating monitors independent of the monitoring strategy. In summary, our contributions are the following:

- a framework for monitoring hyperproperties by a central monitor that has access to all locations (Section 3) and a decentalized monitoring set-up for hyperproperties, with monitors that communicate (Section 4);
- a synthesis function that returns a correct centralized monitor for every formula without least fixed points (Section 3);

- a synthesis function that returns a correct (decentralized) choreography of communicating monitors for every formula without least fixed points that has no location quantifier within a fixed point operator (Section 4); and
- a methodology to prove the correctness of a synthesis of communicating monitors, by establishing a list of desirable properties and relating the behavior of the decentralized monitors to that of the corresponding centralized monitor (Definition 4.5 and Theorem 4.6).

Omitted proofs, due to space constraints, can be found in [2].

2 The Model and the Logic

Let Act be a finite set of actions with at least two elements[2], ranged over by a, b; the set of (infinite) traces over Act is $\mathsf{Trc} = \mathsf{Act}^\omega$, ranged over by t. Given a finite and non-empty set of locations $\mathcal{L}$ ranged over by ℓ, a hypertrace T on $\mathcal{L}$ is a function from $\mathcal{L}$ to Trc; the set of hypertraces on $\mathcal{L}$ is denoted by $\mathsf{HTrc}_\mathcal{L}$. $\mathcal{L}$ and Act are fixed throughout this paper. A hypertrace describes a (distributed) system with $|\mathcal{L}|$ users, and every user is located at a unique location chosen from $\mathcal{L}$. A system behavior is captured by a hypertrace T on $\mathcal{L}$, mapping every user to the trace they perform.

For $t, t' \in \mathsf{Trc}$, we write $t \xrightarrow{a} t'$ whenever $t = at'$. Let $A : \mathcal{L} \to \mathsf{Act}$; for $T, T' \in \mathsf{HTrc}_\mathcal{L}$, we write $T \xrightarrow{A} T'$ whenever $T(\ell) \xrightarrow{A(\ell)} T'(\ell)$, for every $\ell \in \mathcal{L}$. Notice that, for each T, there is a *unique* pair A and T' such that $T \xrightarrow{A} T'$: more precisely, for every $\ell \in \mathcal{L}$, we have that $A(\ell) = a$ and $T'(\ell) = t'$, whenever $T(\ell) = at'$. We denote the A and T' just defined by $hd(T)$ and $tl(T)$ respectively. For a partial function $f : D \rightharpoonup E$ (where D and E are sets ranged over by d and e, respectively), we denote by $\mathsf{dom}(f)$ the set $\{d \in D \mid f(d) \text{ is defined}\}$ and by $\mathsf{rng}(f)$ the set $\{e \mid \exists d \in \mathsf{dom}(f).\ f(d) = e\}$. Notation $f[d \mapsto e]$ denotes the (partial) function mapping d to e and behaving like f otherwise.

2.1 The Logic Hyper-recHML

We consider Hyper-recHML as the logic to specify *hyperproperties*. We assume two disjoint and countably infinite sets Π and V of *location variables* and *recursion variables*, ranged over by π and x, respectively. Formulas of Hyper-recHML are constructed as follows:

$$\varphi ::= \mathsf{tt} \mid \mathsf{ff} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \max x.\varphi \mid \min x.\varphi \mid x \mid \exists \pi.\varphi \mid \forall \pi.\varphi \mid \pi = \pi \mid \pi \neq \pi \mid [a_\pi]\varphi \mid \langle a_\pi \rangle \varphi$$

Apart from the basic boolean constructs, we include the greatest and and least fixed-point operators to describe unbounded and/or infinite behaviors in a finitary manner,[3] existential/universal quantifiers and equality/inequality tests on location variables, and the usual Hennessy-Milner modalities where $[a_\pi]$ stands for 'necessarily after a at the location bound to π', and $\langle a_\pi \rangle$ denotes 'possibly after a at the location bound to π'. A formula is said to be *guarded* if every recursion variable appears within the scope of a modality within its fixed-point binding. All formulas are assumed to be guarded (without loss of expressiveness [41]). We write $\mathsf{FVloc}(\varphi)$ to denote the free location variables of φ, and $\mathsf{FVrec}(\varphi)$ for the free recursion variables.

▶ Remark 2.1. We consider formulas where bound location variables are all pairwise distinct (and different from the free variables); hence, the formula $\forall \pi.[a_\pi]\exists \pi.\varphi$ denotes the formula

[2] When Act is a singleton, every property in the logic becomes equivalent to true or false.

[3] In LTL, this behavior is captured by the 'Until' and 'Release' operators, but these are less expressive than fixed-points; see [7].

$$
\begin{array}{ll}
\llbracket \mathsf{tt} \rrbracket^{\rho}_{\sigma} = \mathsf{HTrc}_{\mathcal{L}} \qquad \llbracket \mathsf{ff} \rrbracket^{\rho}_{\sigma} = \emptyset & \llbracket x \rrbracket^{\rho}_{\sigma} = \rho(x) \\
\llbracket \varphi \wedge \varphi' \rrbracket^{\rho}_{\sigma} = \llbracket \varphi \rrbracket^{\rho}_{\sigma} \cap \llbracket \varphi' \rrbracket^{\rho}_{\sigma} & \llbracket \varphi \vee \varphi' \rrbracket^{\rho}_{\sigma} = \llbracket \varphi \rrbracket^{\rho}_{\sigma} \cup \llbracket \varphi' \rrbracket^{\rho}_{\sigma} \\
\llbracket \max x.\psi \rrbracket^{\rho}_{\sigma} = \bigcup \{ S \mid S \subseteq \llbracket \psi \rrbracket^{\rho[x \mapsto S]}_{\sigma} \} & \llbracket \min x.\psi \rrbracket^{\rho}_{\sigma} = \bigcap \{ S \mid S \supseteq \llbracket \psi \rrbracket^{\rho[x \mapsto S]}_{\sigma} \} \\
\llbracket \exists \pi.\varphi \rrbracket^{\rho}_{\sigma} = \bigcup\limits_{\ell \in \mathcal{L}} \llbracket \varphi \rrbracket^{\rho}_{\sigma[\pi \mapsto \ell]} & \llbracket \forall \pi.\varphi \rrbracket^{\rho}_{\sigma} = \bigcap\limits_{\ell \in \mathcal{L}} \llbracket \varphi \rrbracket^{\rho}_{\sigma[\pi \mapsto \ell]} \\
\llbracket \pi = \pi' \rrbracket^{\rho}_{\sigma} = \begin{cases} \mathsf{HTrc}_{\mathcal{L}} & \text{if } \sigma(\pi) = \sigma(\pi') \\ \emptyset & \text{otherwise} \end{cases} & \llbracket \pi \neq \pi' \rrbracket^{\rho}_{\sigma} = \begin{cases} \mathsf{HTrc}_{\mathcal{L}} & \text{if } \sigma(\pi) \neq \sigma(\pi') \\ \emptyset & \text{otherwise} \end{cases} \\
\multicolumn{2}{l}{\llbracket [a_{\pi}]\varphi \rrbracket^{\rho}_{\sigma} = \{ T \mid \mathit{hd}(T)(\sigma(\pi)) = a \ \text{ implies } \ \mathit{tl}(T) \in \llbracket \varphi \rrbracket^{\rho}_{\sigma} \}} \\
\multicolumn{2}{l}{\llbracket \langle a_{\pi} \rangle \varphi \rrbracket^{\rho}_{\sigma} = \{ T \mid \mathit{hd}(T)(\sigma(\pi)) = a \ \wedge \ \mathit{tl}(T) \ \in \llbracket \varphi \rrbracket^{\rho}_{\sigma}) \}}
\end{array}
$$

Table 1 The semantics of Hyper-recHML.

$\forall \pi.[a_\pi]\exists \pi'.(\varphi\{^{\pi'}/_{\pi}\})$, where $\varphi\{^{\pi'}/_{\pi}\}$ stands for the capture-avoiding substitution of π' for π in φ. A similar notation for other kinds of substitutions is used throughout the paper. ◀

The semantics of a Hyper-recHML formula φ is defined over $\mathsf{HTrc}_{\mathcal{L}}$ by exploiting two partial functions: $\rho\colon V \rightharpoonup 2^{\mathsf{HTrc}_{\mathcal{L}}}$, which assigns a set of hypertraces on $\mathcal{L}$ to all free recursion variables of φ, and $\sigma\colon \Pi \rightharpoonup \mathcal{L}$, which assigns a location to all free location variables of φ. In what follows, we tacitly assume that the free recursion and location variables in a formula φ are always included in $\mathsf{dom}(\rho)$ and $\mathsf{dom}(\sigma)$, respectively.

The semantics for formulas in Hyper-recHML is given through the function $\llbracket - \rrbracket^{\rho}_{\sigma}$ as shown in Table 1. A formula $\langle a_\pi \rangle \varphi$ holds true at hypertrace T if the trace in T at the location bound to π starts with an a and $\mathit{tl}(T)$ satisfies φ; by contrast, a formula $[a_\pi]\varphi$ can also hold true if the trace in T at the location associated to π does not start with an a. Whenever φ is *closed* (i.e., without any free variable), the semantics is given by $\llbracket \varphi \rrbracket^{\emptyset}_{\emptyset}$, where $\emptyset$ denotes the partial function with empty domain. Notationally, we shall simply write $\llbracket \varphi \rrbracket$ instead of $\llbracket \varphi \rrbracket^{\emptyset}_{\emptyset}$. We say that T satisfies the closed formula φ if $T \in \llbracket \varphi \rrbracket$.

▶ **Example 2.2.** For example, consider the set of actions $\{a, b\}$; then, the hyperproperty

$$\varphi_a = \forall \pi. \max x.\big(\langle b_\pi \rangle x \ \vee \ \exists \pi'.(\pi' \neq \pi \ \wedge \ \langle a_{\pi'} \rangle x)\big) \tag{1}$$

is a consensus-type property stating that, at every position of every trace, whenever there is an a there is another trace that also has a. Using the semantic definition of the logic, it is not hard to see that the hypertrace T_1 over the set of locations $\{\ell_1, \ell_2, \ell_3\}$ that maps ℓ_1 to a^ω, ℓ_2 to ba^ω and ℓ_3 to $(ba)^\omega$ does not satisfy the property φ_a: what breaks the property is the first position. On the other hand, the hypertrace T_2 that maps ℓ_1 to a^ω, ℓ_2 to $(ab)^\omega$ and ℓ_3 to $(ba)^\omega$ does satisfy φ_a because at each position there are two traces that exhibit an a. ◀

2.2 On the Expressiveness of Hyper-recHML

The logic Hyper-recHML adapts linear-time μHML [44] to express properties of hypertraces, just as HyperLTL and HyperCTL* [26] are variations on LTL [47] and CTL* [28], respectively, interpreted over hypertraces. It is well known that μHML is more expressive than LTL and CTL* [52]. It is, therefore, natural to wonder whether Hyper-recHML can express properties that cannot be described using HyperLTL and HyperCTL*.

We claim that the strictness of the inclusion of LTL in μHML is preserved for their hyper-extensions. To justify our claim, we present two arguments to demonstrate that Hyper-recHML is more expressive than HyperLTL, which rely on classic results on the inexpressiveness of LTL, the embedding of LTL in μHML, and the ability of Hyper-recHML to quantify over traces more liberally than HyperLTL.

First, we recall that Wolper showed in [52] that the property "event a occurs at all even positions in a trace" cannot be expressed in LTL (see [52, Corollary 4.2] that is based on Theorem 4.1 in that reference). We will refer to this property as φ_e, where "e" stands for even, and adapt it to a hypertrace setting.

▶ **Example 2.3.** Let φ_{h_e} be the hyperproperty on the set of actions $\{a, b\}$ that results from adding an existential trace quantifier $\exists\pi$ at the beginning of φ_e, and replacing all modalities with π-indexed ones:

$$\varphi_{h_e} = \exists\pi.\max x.\big([a_\pi]\langle a_\pi\rangle x \wedge [b_\pi]\langle a_\pi\rangle x\big) \qquad (2)$$

This is a liveness property that describes the periodicity of events; when evaluated over singleton hypertraces, it coincides with the evaluation of φ_e. ◀

The hyperproperty φ_{h_e} defined above can be used to prove the following result.

▶ **Proposition 2.4.** *Hyper-*rec*HML is more expressive than HyperLTL.*

The second witness to the fact that Hyper-recHML is more expressive than HyperLTL is the possibility to use quantifiers in any part of a formula. For example, the hyperproperty φ_a defined in (1) can potentially spawn an unbounded number of quantifiers, by unfolding the recursion when encountering a events.

▶ **Proposition 2.5.** *Hyper-*rec*HML is more expressive than HyperCTL*.*

We shall see later on that part of this additional expressiveness of Hyper-recHML is present in the fragments for which we synthesize monitors.

3 Centralized Monitoring

The set of centralized monitors CMon is given by the following grammar:

$$\mathsf{CMon} \ni m ::= \mathsf{yes} \mid \mathsf{no} \mid \mathsf{end} \mid a_\ell.m \mid m + m \mid m \oplus m \mid m \otimes m \mid \mathsf{rec}\ x.m \mid x$$

Notationally, we denote with $\odot$ any of $\otimes$ and $\oplus$, and use v to range over the verdicts $\{\mathsf{yes}, \mathsf{no}, \mathsf{end}\}$. The operational semantics of centralized monitors is given in Table 2. Notice that monitors that wait for an action at some location (as prescribed by writing a_ℓ) and do not see that action therein (as stated by A) stop their monitoring activity, by reporting end.

Monitors can yield *verdicts* at any point of their computation. This is represented by the judgement $\Rrightarrow$, whose intended use is to evaluate monitors and reach a verdict, whenever possible. The rules are given in Table 3; as one may expect, verdict evaluation is non-deterministic, due to the presence of $+$. Also notice that there can be multiple ways to infer the same verdict for the same monitor: e.g., for yes $\oplus$ no we can either use the third or the (symmetric version of the) fourth rule from the first line of Table 3. However, the inferred value is of course the same (i.e., yes, in the previous situation).

We instrument a monitor m on a hypertrace T based on the rules of Table 4. As usual, we write $\rightarrowtail^*$ for the reflexive-transitive closure of $\rightarrowtail$.

$$v \xrightarrow{A} v \qquad \frac{A(\ell) = a}{a_\ell.m \xrightarrow{A} m} \qquad \frac{A(\ell) \neq a}{a_\ell.m \xrightarrow{A} \mathsf{end}} \qquad \frac{m\{^{\mathsf{rec}\ x.m}/_x\} \xrightarrow{A} m'}{\mathsf{rec}\ x.m \xrightarrow{A} m'} \qquad \frac{m \xrightarrow{A} m'}{m + n \xrightarrow{A} m'}$$

$$\frac{n \xrightarrow{A} n'}{m + n \xrightarrow{A} n'} \qquad \frac{m \xrightarrow{A} m' \qquad n \xrightarrow{A} n'}{m \odot n \xrightarrow{A} m' \odot n'}$$

Table 2 The operational semantics for centralized monitors, where $\odot \in \{\otimes, \oplus\}$.

$$v \Rrightarrow v \qquad \frac{\begin{array}{c} m \Rrightarrow \mathsf{end} \\ n \Rrightarrow \mathsf{end} \end{array}}{m \odot n \Rrightarrow \mathsf{end}} \qquad \frac{m \Rrightarrow \mathsf{yes}}{m \oplus n \Rrightarrow \mathsf{yes}} \qquad \frac{m \Rrightarrow \mathsf{no}}{m \otimes n \Rrightarrow \mathsf{no}}$$

$$\frac{m \Rrightarrow v}{m + n \Rrightarrow v} \qquad \frac{\begin{array}{c} m \Rrightarrow \mathsf{no} \\ n \Rrightarrow v \end{array}}{m \oplus n \Rrightarrow v} \qquad \frac{\begin{array}{c} m \Rrightarrow \mathsf{yes} \\ n \Rrightarrow v \end{array}}{m \otimes n \Rrightarrow v} \qquad \frac{m\{^{\mathsf{rec}\ x.m}/_x\} \Rrightarrow v}{\mathsf{rec}\ x.m \Rrightarrow v}$$

Table 3 Verdict evaluation for centralized monitors (up to commutativity of $+$, $\otimes$, and $\oplus$).

$$\frac{m \xrightarrow{A} m' \qquad T \xrightarrow{A} T'}{m \triangleright T \rightarrowtail m' \triangleright T'}$$

$$\frac{m \Rrightarrow v}{m \triangleright T \rightarrowtail v}$$

Table 4 The instrumentation rules for centralized monitors.

3.1 From Formulas to Centralized Monitors

We derive monitors for the subset of formulas without least fixed-points, denoted with Hyper-maxHML. More precisely, given a formula φ, we want to derive a monitor that, when monitoring a hypertrace T, returns no if and only if T does not belong to the semantics of φ; furthermore, if it returns yes, then T belongs to the semantics of φ. All regular properties of infinite traces that can be monitored for violations with the aforementioned guarantees can be expressed without using least fixed-point operators (see the maximality results presented in [5, Proposition 4.18] and [7, Theorem 5.2] in the setting of logics interpreted over infinite traces). Intuitively, we use least fixed-points to describe liveness properties, whose violation does not have a finite witness in general.

The definition of the synthetized monitor is given by induction on φ. This definition is parametrized by a partial function σ, assigning a location to all the free location variables of φ; when φ is closed, we consider $\mathsf{cm}_\emptyset(\varphi)$. The formal definition is given in Table 5. The interesting cases are for the quantifiers (that are treated as conjunctions and disjunctions, respectively) and for the modal operators.

▶ **Example 3.1.** Let $\mathcal{L} = \{1, 2\}$ and $\mathsf{Act} = \{a, b\}$, and consider the formula (2). The monitor synthesis in Table 5 produces the following monitor m when applied to that formula:

$$m = \bigoplus_{\ell \in \{1,2\}} \mathsf{rec}\ x.((a_\ell.(a_\ell.x + b_\ell.\mathsf{no}) + b_\ell.\mathsf{yes}) \otimes (b_\ell.(a_\ell.x + b_\ell.\mathsf{no}) + a_\ell.\mathsf{yes})).$$

When monitor m is instrumented with the hypertrace T mapping location 1 to a^ω and location 2 to $(ab)^\omega$, the verdict no cannot be reached: indeed, T satisfies the formula φ since the trace at location 1 has a at all positions. On the other hand, when m is instrumented with the hypertrace T' mapping location 1 to b^ω and location 2 to $(ab)^\omega$, the no verdict is reached after the monitor has observed the first two actions at locations 1 and 2; this is in line with the fact that T' does not satisfy φ_{h_e}. ◀

$$\mathsf{cm}_\sigma(\mathsf{tt}) = \mathsf{yes} \qquad \mathsf{cm}_\sigma(\mathsf{ff}) = \mathsf{no} \qquad\qquad \mathsf{cm}_\sigma(x) = x \qquad \mathsf{cm}_\sigma(\max x.\varphi) = \mathsf{rec}\ x.\mathsf{cm}_\sigma(\varphi)$$

$$\mathsf{cm}_\sigma(\varphi \wedge \varphi') = \mathsf{cm}_\sigma(\varphi) \otimes \mathsf{cm}_\sigma(\varphi') \qquad\qquad \mathsf{cm}_\sigma(\varphi \vee \varphi') = \mathsf{cm}_\sigma(\varphi) \oplus \mathsf{cm}_\sigma(\varphi')$$

$$\mathsf{cm}_\sigma(\forall\pi.\varphi) = \textstyle\bigotimes_{\ell\in\mathcal{L}} \mathsf{cm}_{\sigma[\pi\mapsto\ell]}(\varphi) \qquad\qquad \mathsf{cm}_\sigma(\exists\pi.\varphi) = \textstyle\bigoplus_{\ell\in\mathcal{L}} \mathsf{cm}_{\sigma[\pi\mapsto\ell]}(\varphi)$$

$$\mathsf{cm}_\sigma(\pi = \pi') = \begin{cases} \mathsf{yes} \text{ if } \sigma(\pi) = \sigma(\pi') \\ \mathsf{no} \text{ otherwise} \end{cases} \qquad\qquad \mathsf{cm}_\sigma(\pi \neq \pi') = \begin{cases} \mathsf{yes} \text{ if } \sigma(\pi) \neq \sigma(\pi') \\ \mathsf{no} \text{ otherwise} \end{cases}$$

$$\mathsf{cm}_\sigma([a_\pi]\varphi) = a_{\sigma(\pi)}.\mathsf{cm}_\sigma(\varphi) + \textstyle\sum_{b\neq a} b_{\sigma(\pi)}.\mathsf{yes} \qquad \mathsf{cm}_\sigma(\langle a_\pi\rangle\varphi) = a_{\sigma(\pi)}.\mathsf{cm}_\sigma(\varphi) + \textstyle\sum_{b\neq a} b_{\sigma(\pi)}.\mathsf{no}$$

Table 5 Centralized monitor synthesis.

The main results of this section are that the centralized monitors synthesized from formulas report sound verdicts and their verdicts are complete for formula violations. We refer the reader to [7] for a discussion on notions of correctness for monitors and the significance of soundness and violation-completeness. The proofs can be found in [2].

▶ **Theorem 3.2** (Soundness). *Let $\varphi \in$ Hyper-maxHML be a closed formula and $T \in \mathsf{HTrc}_\mathcal{L}$. If $\mathsf{cm}_\emptyset(\varphi) \triangleright T \rightarrowtail^* \mathsf{no}$, then $T \notin [\![\varphi]\!]$; if $\mathsf{cm}_\emptyset(\varphi) \triangleright T \rightarrowtail^* \mathsf{yes}$, then $T \in [\![\varphi]\!]$.*

▶ **Theorem 3.3** (Violation Completeness). *Let $\varphi \in$ Hyper-maxHML be a closed formula and $T \in \mathsf{HTrc}_\mathcal{L}$. If $T \notin [\![\varphi]\!]$, then $\mathsf{cm}_\emptyset(\varphi) \triangleright T \rightarrowtail^* \mathsf{no}$.*

4 Decentralized Monitoring

When verifying a distributed system, having a central authority that performs any type of runtime verification is a strong assumption, as it reduces the appeal of distribution. Thus, we study to what extent hyperproperties can be monitored by decentralized monitors.

We associate monitors to locations, denoted by ℓ, and monitors associated to ℓ monitor only actions required to happen at ℓ, thus allowing the processing of events to happen locally. This imposes some form of coordination between monitors at different locations. For this reason, we introduce the possibility for monitors to communicate.

We define a communication alphabet Com, ranged over by c, over some finite alphabet of communication constants Con (that contains Act), ranged over by γ, as

$$\mathsf{Com} \ni c \ ::= \ (!G, \gamma) \ \mid \ (?G, \gamma),$$

where $G \subseteq \mathcal{L}$ and $\gamma \in \mathsf{Con}$. We have a communication action $(!G, \gamma)$ for sending γ to group G (multicast communication), and one $(?G, \gamma)$ for receiving γ from any monitor from the set G. Point-to-point communication can be represented by taking singleton sets for G.

The syntax of decentralized monitors is given by the following grammar:

$$\mathsf{DMon} \ni M ::= [m]_\ell \ \mid \ M \vee M \ \mid M \wedge M$$
$$\mathsf{LMon} \ni m ::= \mathsf{yes} \mid \mathsf{no} \mid \mathsf{end} \mid a.m \mid c.m \mid m + m \mid m \oplus m \mid m \otimes m \mid \mathsf{rec}\ x.m \mid x$$

Monitor $[m]_\ell$ denotes that m monitors the trace located at location ℓ, so, it is ‘localized’ at ℓ (this justifies the name LMon). Monitors assigned to the same trace run in parallel and observe identical events; contrary to [1], monitors assigned to different traces are no

$$a.m \xrightarrow{a} m \qquad \frac{\ell \in G}{(?G,\gamma).m \xrightarrow{(?\ell,\gamma)} m} \qquad (!G,\gamma).m \xrightarrow{(!G,\gamma)} m \qquad v \xrightarrow{a} v$$

$$\frac{m\{^{\mathsf{rec}\ x.m}/_{x}\} \xrightarrow{\lambda} m'}{\mathsf{rec}\ x.m \xrightarrow{\lambda} m'} \qquad \frac{m \xrightarrow{a} m' \qquad n \xrightarrow{a} n'}{m \odot n \xrightarrow{a} m' \odot n'} \qquad \frac{m \xrightarrow{(?\ell,\gamma)} m' \qquad n \xrightarrow{(?\ell,\gamma)} n'}{m \odot n \xrightarrow{(?\ell,\gamma)} m' \odot n'}$$

$$\frac{m \xrightarrow{\lambda} m'}{m + n \xrightarrow{\lambda} m'} \qquad \frac{m \xrightarrow{(!G,\gamma)} m'}{m \odot n \xrightarrow{(!G,\gamma)} m' \odot n} \qquad \frac{m \xrightarrow{(?\ell,\gamma)} m' \qquad n \not\xrightarrow{(?\ell,\gamma)}}{m \odot n \xrightarrow{(?\ell,\gamma)} m' \odot n}$$

Table 6 The operational semantics for decentralized local monitors (up to commutativity of $+$, $\otimes$ and $\oplus$), where we let λ denote either a, $(!G,\gamma)$ or $(?\ell,\gamma)$ for $\ell \in \mathcal{L}$, $G \subseteq \mathcal{L}$.

$$\frac{m \xrightarrow{(!G,\gamma)} m'}{[m]_\ell \xrightarrow{\ell:(!G,\gamma)} [m']_\ell} \qquad \frac{m \xrightarrow{(?\ell',\gamma)} m' \qquad \ell \in G}{[m]_\ell \overset{G\,:\,(?\ell',\gamma)}{\rightsquigarrow} [m']_\ell}$$

$$\frac{m \not\xrightarrow{(?\ell',\gamma)}}{[m]_\ell \overset{G\,:\,(?\ell',\gamma)}{\rightsquigarrow} [m]_\ell} \qquad \frac{\ell \notin G}{[m]_\ell \overset{G\,:\,(?\ell',\gamma)}{\rightsquigarrow} [m]_\ell}$$

$$\frac{M \overset{G\,:\,(?\ell,\gamma)}{\rightsquigarrow} M' \qquad N \overset{G\,:\,(?\ell,\gamma)}{\rightsquigarrow} N'}{M \diamond N \overset{G\,:\,(?\ell,\gamma)}{\rightsquigarrow} M' \diamond N'} \diamond \in \{\wedge, \vee\}$$

$$\frac{M \xrightarrow{\ell:(!G,\gamma)} M' \qquad N \overset{G\,:\,(?\ell,\gamma)}{\rightsquigarrow} N'}{M \diamond N \xrightarrow{\ell:(!G,\gamma)} M' \diamond N'} \diamond \in \{\wedge, \vee\}$$

Table 7 Operational semantics for communication of $M \in \mathsf{DMon}$ (up to commutativity of $\wedge$, $\vee$).

$$\frac{A(\ell) = a \qquad m \xrightarrow{a} m'}{[m]_\ell \xrightarrow{A} [m']_\ell}$$

$$\frac{A(\ell) = a \qquad m \not\xrightarrow{a} \qquad m \not\xrightarrow{c}}{[m]_\ell \xrightarrow{A} [\mathsf{end}]_\ell}$$

$$\frac{M \xrightarrow{A} M' \qquad N \xrightarrow{A} N'}{M \diamond N \xrightarrow{A} M' \diamond N'} \diamond \in \{\wedge, \vee\}$$

Table 8 Operational semantics for actions of $M \in \mathsf{DMon}$ (up to commutativity of $\wedge$, $\vee$).

longer completely isolated from each other, but can now communicate, which is the main new feature of the decentralized set-up.

The operational rules for $m \in \mathsf{LMon}$ are given in Table 6. Notice that, when we have parallel monitors, only one of them at a time can send; by contrast, all those that can receive from some location ℓ are forced to do so.

For $M \in \mathsf{DMon}$, the operational semantics can be found in Table 7 (the rules concerning communication) and Table 8 (the rules concerning action steps). The operational semantics in Table 7 defines multicast, where a monitor located at ℓ sends a message to group G and every monitor at a location in G that can receive from ℓ does so; every monitor that cannot, or that is not in G, does not change its state. The first four rules capture the judgment for inferring when all components of a monitor which are able to receive a certain γ sent from a location do so. Intuitively, ℓ is the location from which message γ was sent to group G, and $M \overset{G\,:\,(?\ell,\gamma)}{\rightsquigarrow} N$ indicates that every monitor in M located at a location in G that can receive γ from ℓ indeed has received γ and transitioned appropriately in N. The last two rules then actually define communication. In particular, the last rule in Table 7 implements multicast by stipulating that the outcome of the synchronization between a send action $\ell : (!G,\gamma)$

$$\frac{m \Rrightarrow v}{[m]_\ell \Rrightarrow v} \qquad \frac{M \Rrightarrow \mathsf{end} \quad N \Rrightarrow \mathsf{end}}{M \diamond N \Rrightarrow \mathsf{end}}$$

$$\frac{M \Rrightarrow \mathsf{no}}{M \wedge N \Rrightarrow \mathsf{no}} \qquad \frac{M \Rrightarrow \mathsf{yes} \quad N \Rrightarrow v}{M \wedge N \Rrightarrow v}$$

$$\frac{M \Rrightarrow \mathsf{yes}}{M \vee N \Rrightarrow \mathsf{yes}} \qquad \frac{M \Rrightarrow \mathsf{no} \quad N \Rrightarrow v}{M \vee N \Rrightarrow v}$$

Table 9 The verdict combination rules for decentralized monitors (up to commutativity of $\wedge$ and $\vee$, ranged over by $\diamond$).

$$\frac{M \xrightarrow{A} M' \qquad T \xrightarrow{A} T'}{M \triangleright T \rightarrowtail M' \triangleright T'}$$

$$\frac{M \xrightarrow{\ell:(!G,\gamma)} M'}{M \triangleright T \rightarrowtail M' \triangleright T}$$

$$\frac{M \Rrightarrow v}{M \triangleright T \rightarrowtail v}$$

Table 10 The evolution of a decentralized monitor instrumented on a hypertrace.

and a receive one of the form $G : (?\ell, \gamma)$ is the send action itself, which can be received by other monitors at locations in G in a larger monitor of which $M \diamond N$ is a sub-term. We note, in passing, that monitors $M \in \mathsf{DMon}$ are 'input-enabled': for each M, G, ℓ and γ, there is always some M' such that $M \overset{G\,:\,(?\ell,\gamma)}{\rightsquigarrow} M'$. So the last rule in Table 7 (and its symmetric version) can always be applied when the send transition in its premise is available.

Monitors can also locally observe an action, as prescribed by a location-to-action function A; the rules are given in Table 8. Monitors at the same location observe the same action. If a monitor cannot take the action prescribed by A at its location, the monitor becomes end, as stipulated by the second rule given in Table 8. Note that it is not sufficient to trigger that rule when m cannot exhibit action $A(\ell)$: we also require that m cannot communicate. Note that the inability of m to exhibit action $A(\ell)$ is not sufficient to trigger that rule: we also require that m cannot communicate. Intuitively, this is because monitors exhibit an 'alternating' behavior in which they observe the next action produced by a system hypertrace and then embark in a sequence of communications with other monitors to inform them of what they observed. As will be made clear in our definition of a weak bisimulation relation presented in Definition 4.1, such communications are interpreted as internal actions in monitor behavior. Therefore, the inability of some monitor $[m]_\ell$ to perform action $A(\ell)$ can only be gauged in 'stable states'—that is, monitor states in which no communication is possible. This design choice is akin to that underlying the definition of refusal testing presented in [46] and of the stable-failures model for (Timed) CSP defined in [49, 50], where the inability of a process to perform some action can only be determined in states that afford no internal computation steps.

Verdict evaluation for $M \in \mathsf{DMon}$ is defined in Table 9 and relies on that for $m \in \mathsf{CMon}$ provided in Table 3. Finally, given a decentalized monitor M and a hypertrace T, the instrumentation of the monitor on the trace is described by the rules of Table 10. As before, we denote with $\rightarrowtail^*$ the reflexive transitive closure of $\rightarrowtail$.

4.1 Synthesizing Decentralized Monitors Correctly

In this section we describe how to synthesize decentralized monitors 'correctly' from formulas, i.e. such that their behavior corresponds to that of the corresponding centralized monitors. The advantage of this approach is that it simplifies the proof that monitors synthesized via a 'correct' decentralized synthesis function are sound and violation-complete, by utilizing the correspondence to centralized monitors. Moreover, it identifies desirable properties of a 'correct' decentralized synthesis function that can guide the development of further automated decentralized-monitor synthesis algorithms.

We first define the correspondence between centralized and decentralized monitors and show that this correspondence is sufficient to obtain soundness and violation-completeness in the decentralized setting from the corresponding results in the centralized setting (Theorems 3.2 and 3.3). In the remainder of the section, given a synthesis function which takes as inputs a formula φ and a mapping σ from location variables to locations, and outputs a monitor $\mathcal{M}_\sigma(\varphi) \in \mathsf{DMon}$, we specify criteria that allow us to derive this correspondence.

We write $M \twoheadrightarrow M'$ to denote the existence of an integer $h > 0$ and of h monitors $M_1, \ldots, M_h$, locations $\ell_1, \ldots, \ell_{h-1}$ and communication actions $c_1, \ldots, c_{h-1}$ such that $M_1 = M$, $M_h = M'$, and $M_i \xrightarrow{\ell_i : c_i} M_{i+1}$ (for every $i = 1, \ldots, h-1$). By definition of $\rightarrow$ on communicating monitors, each c_i is $(!G_i, \gamma_i)$, for some $G_i \subseteq \mathcal{L}$ and $\gamma_i \in \mathsf{Con}$. Similarly, at the level of local monitors we write $m \twoheadrightarrow m'$ to denote the existence of an integer $h > 0$, of local monitors $m_1, \ldots, m_h$ and of $c_1, \cdots c_h \in \{(!G, \gamma), (?\ell, \gamma) \mid G \subseteq \mathcal{L}, \ell \in \mathcal{L}, \gamma \in \mathsf{Con}\}$ such that $m_1 = m$, $m_h = m'$ and $m_i \xrightarrow{c_i} m_{i+1}$.

The correspondence between the centralized and the decentralized monitors is characterized as a weak bisimulation:

▶ **Definition 4.1.** *A binary relation $\mathcal{R}$ over* $\mathsf{DMon} \times \mathsf{CMon}$ *is a weak bisimulation if and only if, whenever $M \mathcal{R} m$, it holds that:*

1. *$\exists M' \in \mathsf{DMon}$ such that $M \twoheadrightarrow M'$ and $M' \Rrightarrow v$ if and only if $m \Rrightarrow v$.*
2. *If $M \xrightarrow{A} M'$ then $\exists m' \in \mathsf{CMon}$ such that $m \xrightarrow{A} m'$ and $M' \mathcal{R} m'$.*
3. *If $M \xrightarrow{c} M'$ then $M' \mathcal{R} m$, where $c = \ell : (!G, \gamma)$ for some $\ell \in \mathcal{L}$, $G \subseteq \mathcal{L}$, $\gamma \in \mathsf{Con}$.*
4. *If $m \xrightarrow{A} m'$ then there exist M_1, M_2, M' such that $M \twoheadrightarrow M_1 \xrightarrow{A} M_2 \twoheadrightarrow M'$ and $M' \mathcal{R} m'$.*

One of the main features of weak bisimilarity is that, if $\mathcal{M}_\sigma(\varphi)$ and $\mathsf{cm}_\sigma(\varphi)$ are weakly bisimilar, then they report the same verdict when observing any hypetrace T; thus, we obtain violation-completeness and soundness for decentralized monitors from the corresponding results for centralized monitors:

▶ **Corollary 4.2** (Soundness). *Let $T \in \mathsf{HTrc}_\mathcal{L}$, $\varphi \in$ Hyper-maxHML be a closed formula such that $\mathcal{M}_\emptyset(\varphi)$ is defined, and $\mathcal{R}$ a weak bisimulation such that $(\mathcal{M}_\emptyset(\varphi), \mathsf{cm}_\emptyset(\varphi)) \in \mathcal{R}$. If $\mathcal{M}_\emptyset(\varphi) \triangleright T \rightarrowtail^* \mathsf{no}$, then $T \notin [\![\varphi]\!]$; if $\mathcal{M}_\emptyset(\varphi) \triangleright T \rightarrowtail^* \mathsf{yes}$, then $T \in [\![\varphi]\!]$.*

▶ **Corollary 4.3** (Violation Completeness). *Let $T \in \mathsf{HTrc}_\mathcal{L}$, $\varphi \in$ Hyper-maxHML be a closed formula such that $\mathcal{M}_\emptyset(\varphi)$ is defined, and $\mathcal{R}$ a weak bisimulation such that $(\mathcal{M}_\emptyset(\varphi), \mathsf{cm}_\emptyset(\varphi)) \in \mathcal{R}$. If $T \notin [\![\varphi]\!]$, then $\mathcal{M}_\emptyset(\varphi) \triangleright T \rightarrowtail^* \mathsf{no}$.*

We now describe sufficient conditions for any decentralized synthesis function such that there is a weak bisimulation between the centralized and the decentralized monitors synthesized from a formula φ and a location environment σ. Whenever we write $M \xrightarrow{c} N$ for $M, N \in \mathsf{DMon}$, we assume that $c \in \{\ell : (!G, \gamma) \mid \ell \in \mathcal{L}, G \subseteq \mathcal{L}, \gamma \in \mathsf{Con}\}$, as per the labeling of the communication transitions of decentralized monitors. We write $[m]_\ell \in M$, for $M \in \mathsf{DMon}$, if $[m]_\ell$ is one of its constituents: formally, $[m]_\ell \in [m]_\ell$ and, if $[m]_\ell \in M$, then $[m]_\ell \in M \diamond N$ and $[m]_\ell \in N \diamond M$ (recall that $\diamond$ denotes either $\wedge$ or $\vee$). We start by defining when $M \in \mathsf{DMon}$ can(not) communicate:

▶ **Definition 4.4.** *Let $M \in \mathsf{DMon}$. We say $M \in \mathsf{DMon}$ can communicate, if there exists $[m]_\ell \in M$ such that $m \xrightarrow{c} n$ for some $c \in \mathsf{Com}$. Otherwise, we say M cannot communicate.*

▶ **Definition 4.5.** *We say that a monitor synthesis $\mathcal{M}_{\_}(-)$ is* principled *when it satisfies the following conditions, for every formula φ and environment σ such that $\mathcal{M}_\sigma(\varphi)$ is defined:*
Verdict Agreement: *for every verdict v, $\mathsf{cm}_\sigma(\varphi) \Rrightarrow v$ if and only if $\mathcal{M}_\sigma(\varphi) \Rrightarrow v$;*

Verdict Irrevocability: *for every verdict v and $\mathcal{M}_\sigma(\varphi) \xrightarrow{A} M_1 \rightarrow M_2 \rightarrow M$, if $M_2 \Rrightarrow v$, then $M \Rrightarrow v$;*

Reactivity: *for every A, there exists M such that $\mathcal{M}_\sigma(\varphi) \xrightarrow{A} M$;*

Bounded Communication: *for every $\mathcal{M}_\sigma(\varphi) \xrightarrow{A} M \rightarrow M'$, there exists M'' such that $M' \rightarrow M''$ and M'' cannot communicate;*

Processing-Communication Alternation: *for every $\mathcal{M}_\sigma(\varphi) \xrightarrow{A} M \rightarrow M_1$,*

1. *$\mathcal{M}_\sigma(\varphi)$ cannot communicate, and*
2. *$M_1 \xrightarrow{c} M_2$ implies $M_1 \not\xrightarrow{A}$ for every c and A;*

Formula Convergence: *if $\mathcal{M}_\sigma(\varphi) \xrightarrow{A} M \rightarrow M'$, M' cannot communicate, and $\textsc{cm}_\sigma(\varphi) \xrightarrow{A} \textsc{cm}_{\sigma'}(\varphi')$ for some formula φ' and environment σ', then $M' = \mathcal{M}_{\sigma'}(\varphi')$.*

Let $\mathcal{M}_{-}(-)$ be a decentralized synthesis function. We define relation $\mathcal{R}_{\mathcal{M}}$ as follows:

$$\mathcal{R}_{\mathcal{M}} \triangleq \mathcal{R}_1 \cup \mathcal{R}_2$$

$$\mathcal{R}_1 \triangleq \{(\mathcal{M}_\sigma(\varphi), \textsc{cm}_\sigma(\varphi)) \mid \mathsf{FVloc}(\varphi) \subseteq \mathsf{dom}(\sigma)\}$$

$$\mathcal{R}_2 \triangleq \left\{(M', \textsc{cm}_{\sigma'}(\varphi')) \mid \mathsf{FVloc}(\varphi) \subseteq \mathsf{dom}(\sigma) \text{ and } \mathcal{M}_\sigma(\varphi) \xrightarrow{A} M \rightarrow M' \rightarrow \mathcal{M}_{\sigma'}(\varphi')\right\}$$

The crucial property of any principled synthesis function is the following:

▶ **Theorem 4.6.** *For every principled synthesis $\mathcal{M}_{-}(-)$, $\mathcal{R}_{\mathcal{M}}$ is a weak bisimulation.*

4.2 From Formulas to Decentralized Monitors

We now describe how to synthesize decentralized monitors for a fragment of Hyper-maxHML, and show that this synthesis function satisfies Definition 4.5. This allows us to apply Theorem 4.6 and obtain soundness and violation-completeness of these synthesized monitors.

In what follows, we consider formulas from PHyper-recHML, the subset of Hyper-recHML given by the following grammar (see Section 5 for a discussion on the choice of fragment):

$$\varphi ::= \exists\pi.\varphi \mid \forall\pi.\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \psi$$

$$\psi ::= \mathsf{tt} \mid \mathsf{ff} \mid \pi = \pi \mid \pi \neq \pi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \max x.\psi \mid \min x.\psi \mid x \mid [a_\pi]\psi \mid \langle a_\pi \rangle \psi$$

We denote the class of formulas of type ψ with Qf (quantifier free). PHyper-recHML is a subset of Hyper-recHML and thus its semantics over $\mathsf{HTrc}_{\mathcal{L}}$ is the one given in Table 1.

We synthesize decentralized monitors for the fragment of PHyper-recHML only containing formulas of type ψ without diamonds and least fixed-points, which we call PHyper-maxHML. In section 4.3 we also discuss how diamonds can also be added to the picture. The synthesis for decentralized monitors is given in Table 11. First, we derive a monitor belonging to LMon for formulas of type $\psi \in \mathsf{Qf}$; this synthesis function is parametrized by a location $\ell \in \mathcal{L}$ and a partial function σ from Π to $\mathcal{L}$ that is defined for every free location variable in ψ. Then we derive monitors belonging to DMon for formulas of type φ.

Note that, in the definition of $\textsc{dm}_\sigma(\psi)$, $\textsc{cm}_\sigma(\psi)$ is the monitor resulting from the centralized synthesis function defined in Table 5. Intuitively $\textsc{dm}_\sigma(\psi)$ synthesizes a local monitor at each location relevant to ψ, which are the locations associated by σ to the free location variables in ψ. If $\sigma = \emptyset$ (and so ψ does not have any free trace variables), there is no need for communication between locations, and in fact a verdict can be obtained from ψ immediately. This verdict coincides with the one reached in the centralized synthesis.

We observe that the case for $\sigma = \emptyset$ and $\textsc{cm}_\sigma(\psi) \Rrightarrow v$ only applies when ψ is a Boolean combination of tt and ff. Thus, every closed formula φ on which we apply our synthesis

$$\mathrm{Dm}^\ell_\sigma(\mathsf{tt}) = \mathsf{yes} \qquad \mathrm{Dm}^\ell_\sigma(\mathsf{ff}) = \mathsf{no} \qquad \mathrm{Dm}^\ell_\sigma(x) = x \qquad \mathrm{Dm}^\ell_\sigma(\max x.\psi) = \mathsf{rec}\ x.\mathrm{Dm}^\ell_\sigma(\psi)$$

$$\mathrm{Dm}^\ell_\sigma(\psi \wedge \psi') = \mathrm{Dm}^\ell_\sigma(\psi) \otimes \mathrm{Dm}^\ell_\sigma(\psi') \qquad \mathrm{Dm}^\ell_\sigma(\psi \vee \psi') = \mathrm{Dm}^\ell_\sigma(\psi) \oplus \mathrm{Dm}^\ell_\sigma(\psi')$$

$$\mathrm{Dm}^\ell_\sigma([a_\pi]\psi) = \begin{cases} a.(!(\mathsf{rng}(\sigma)\backslash\{\ell\}), a).\mathrm{Dm}^\ell_\sigma(\psi) + \sum\limits_{b \neq a} b.(!(\mathsf{rng}(\sigma)\backslash\{\ell\}), b).\mathsf{yes} & \text{if } \sigma(\pi) = \ell \\ \sum\limits_{b \in \mathsf{Act}} b.\Big((?\{\sigma(\pi)\}, a).\mathrm{Dm}^\ell_\sigma(\psi) \ + \sum\limits_{b \neq a} (?\{\sigma(\pi)\}, b).\mathsf{yes}\Big) & \text{otherwise} \end{cases}$$

$$\mathrm{Dm}^\ell_\sigma(\pi = \pi') = \begin{cases} \mathsf{yes} & \text{if } \sigma(\pi) = \sigma(\pi') \\ \mathsf{no} & \text{otherwise} \end{cases} \qquad \mathrm{Dm}^\ell_\sigma(\pi \neq \pi') = \begin{cases} \mathsf{yes} & \text{if } \sigma(\pi) \neq \sigma(\pi') \\ \mathsf{no} & \text{otherwise} \end{cases}$$

$$\mathrm{DM}_\sigma(\psi) = \begin{cases} \bigvee_{\ell \in \mathsf{rng}(\sigma)} [\mathrm{Dm}^\ell_\sigma(\psi)]_\ell & \text{if } \sigma \neq \emptyset \\ [v]_{\ell_0} & \text{if } \sigma = \emptyset \wedge \mathrm{Cm}_\sigma(\psi) \Rrightarrow v \end{cases}$$

$$\mathrm{DM}_\sigma(\forall \pi.\varphi) = \bigwedge\nolimits_{\ell \in \mathcal{L}} \mathrm{DM}_{\sigma[\pi \mapsto \ell]}(\varphi) \qquad \mathrm{DM}_\sigma(\exists \pi.\varphi) = \bigvee\nolimits_{\ell \in \mathcal{L}} \mathrm{DM}_{\sigma[\pi \mapsto \ell]}(\varphi)$$

$$\mathrm{DM}_\sigma(\varphi \wedge \varphi') = \mathrm{DM}_\sigma(\varphi) \wedge \mathrm{DM}_\sigma(\varphi') \qquad \mathrm{DM}_\sigma(\varphi \vee \varphi') = \mathrm{DM}_\sigma(\varphi) \vee \mathrm{DM}_\sigma(\varphi')$$

Table 11 Decentralized monitor synthesis, where ℓ_0 is any fixed element of $\mathcal{L}$.

1. is trivial, *i.e.* φ is logically equivalent to tt or ff, or
2. is such that every subformula $\psi \in \mathsf{Qf}$ of φ is in the scope of a quantifier.

For non-trivial formulas, the $\sigma = \emptyset$ case for $\mathrm{DM}_\sigma(\psi)$ never applies, and we can ignore it. The decentralized monitor for a closed formula φ is $\mathrm{DM}_\emptyset(\varphi)$.

▶ Remark 4.7. In the first clause of the definition of the synthesis function for box formulas, it might seem superfluous to send a message also when the monitor observes some $b \neq a$. However, this is important to make sure monitors do not deadlock. To see this, consider a synthesis where that definition instead looks like

$$\mathrm{Dm}^\ell_\sigma([a_\pi]\psi) = \begin{cases} a.(!(\mathsf{rng}(\sigma)\backslash\{\ell\}), a).\mathrm{Dm}^\ell_\sigma(\psi) + \sum\limits_{b \neq a} b.\mathsf{yes} & \text{if } \sigma(\pi) = \ell \\ \sum\limits_{b \in \mathsf{Act}} b.(?\{\sigma(\pi)\}, a).\mathrm{Dm}^\ell_\sigma(\psi) & \text{otherwise} \end{cases}$$

Consider $\mathsf{Act} = \{a, b\}$, $\mathcal{L} = \{\ell, \ell'\}$ and some hypertrace T such that $T(\ell) = b.t_1$ and $T(\ell') = b.t_2$ for some traces t_1 and t_2. Now consider $m \otimes n$, where $m = \mathrm{Dm}^\ell_\sigma([a_\pi]\psi)$, $n = \mathrm{Dm}^\ell_\sigma([a_{\pi'}]\psi')$, $\sigma(\pi) = \ell$ and $\sigma(\pi') = \ell'$. For $A(\ell) = A(\ell') = b \neq a$, we then get $m \xrightarrow{A(\ell)} \mathsf{yes}$ and $n \xrightarrow{A(\ell')} (?\{\sigma(\pi')\}, a).\mathrm{Dm}^\ell_\sigma(\psi')$, and monitor $\mathsf{yes} \otimes (?\{\sigma(\pi')\}, a).\mathrm{Dm}^\ell_\sigma(\psi')$ is stuck because the receive action of the monitor $(?\{\sigma(\pi')\}, a).\mathrm{Dm}^\ell_\sigma(\psi')$ has no matching send. It is precisely to avoid these scenarios that we make sure that, for each sending transition, there is a corresponding receiving transition, and a monitor always sends the last action it read to all other locations in the range of the environment σ. ◀

Soundness and violation completeness for the synthesis defined in Table 11 follow from Corollary 4.2 and 4.3 by using Theorem 4.6, once we prove the following key result:

▶ **Theorem 4.8.** *The synthesis function* DM *defined in Table 11 is principled.*

▶ **Example 4.9.** In order to highlight the inter-monitor communication, we consider the following formula

$$\varphi = \exists \pi.\exists \pi'.([a_\pi]\mathsf{ff} \wedge [b_{\pi'}]\mathsf{ff})$$

over $\mathcal{L} = \{1, 2\}$ and $\mathsf{Act} = \{a, b\}$, which states that either both traces start with a, or neither does. By letting $\sigma = [\pi \mapsto \ell, \pi' \mapsto \ell']$, the synthesis for this property gives:

$$\mathrm{D}\mathsf{M}_{\emptyset}(\varphi) = \bigvee_{\ell,\ell' \in \mathcal{L}} \; \bigvee_{\ell'' \in \{\ell,\ell'\}} \left[\mathrm{D}\mathsf{m}^{\ell''}_{\sigma}([a_\pi]\mathsf{ff} \wedge [b_{\pi'}]\mathsf{ff})\right]_{\ell''} \text{ , where}$$

$$\mathrm{D}\mathsf{m}^{\ell''}_{\sigma}([a_\pi]\mathsf{ff} \wedge [b_{\pi'}]\mathsf{ff}) = \begin{cases} (a.(!\emptyset, a).\mathsf{no} + b.(!\emptyset, b).\mathsf{yes}) \otimes & \text{if } \ell = \ell' = \ell'' \\ \quad (b.(!\emptyset, b).\mathsf{no} + a.(!\emptyset, a).\mathsf{yes}) & \\ (a.(!\{\ell'\}, a).\mathsf{no} + b.(!\{\ell'\}, b).\mathsf{yes}) \otimes & \text{if } \ell \neq \ell' \text{ and } \ell'' = \ell \\ \quad (a.((?\{\ell'\}, b).\mathsf{no} + (?\{\ell'\}, a).\mathsf{yes}) + & \\ \quad\; b.((?\{\ell'\}, b).\mathsf{no} + (?\{\ell'\}, a).\mathsf{yes})) & \\ (a.((?\{\ell\}, a).\mathsf{no} + (?\{\ell\}, b).\mathsf{yes}) + & \text{if } \ell \neq \ell' \text{ and } \ell'' = \ell' \\ \; b.((?\{\ell\}, a).\mathsf{no} + (?\{\ell\}, b).\mathsf{yes})) & \\ \otimes\ (b.(!\{\ell\}, b).\mathsf{no} + a.(!\{\ell\}, a).\mathsf{yes}) & \end{cases}$$

◀

4.3 On the Decentralized-Monitor Synthesis for Diamonds

The synthesis of decentralized monitors presented in Table 11 does not deal explicitly with formulas of the form $\langle a_\pi \rangle \psi$. However, it can be applied to those formulas using the observation that $\langle a_\pi \rangle \psi$ is logically equivalent to

$$[a_\pi]\psi \wedge \textstyle\bigwedge_{b \neq a} [b_\pi]\mathsf{ff}. \tag{3}$$

To showcase this, we present an example of the decentralized synthesis applied on Wolper's property (φ_{h_e}) from Example 2.3, which makes use of diamond modalities.

▶ **Example 4.10.** Recall φ_{h_e} from (2); expressed here as $\exists \pi.\psi$, with

$$\psi = \max x.(\psi_1 \wedge \psi_2) \qquad \psi_1 = [a_\pi]\langle a_\pi \rangle x \qquad \psi_2 = [b_\pi]\langle a_\pi \rangle x$$

Let $\mathcal{L} = \{1, 2\}$ and $\mathsf{Act} = \{a, b\}$. The synthesis is applied thus:

$$\mathrm{D}\mathsf{M}_{\emptyset}(\varphi) = \textstyle\bigvee_{\ell \in \mathcal{L}} \left[\mathsf{rec}\ x.\ \left(m^{\ell}_{[\pi \mapsto \ell]}(\psi_1) \otimes m^{\ell}_{[\pi \mapsto \ell]}(\psi_2)\right)\right]_{\ell}$$

with

$$\begin{aligned} m^{\ell}_{[\pi \mapsto \ell]}(\psi_1) &= a.(!\emptyset, a).m^{\ell}_{[\pi \mapsto \ell]}(\langle a_\pi \rangle x) + b.(!\emptyset, b).\mathsf{yes} \\ m^{\ell}_{[\pi \mapsto \ell]}(\psi_2) &= b.(!\emptyset, b).m^{\ell}_{[\pi \mapsto \ell]}(\langle a_\pi \rangle x) + a.(!\emptyset, a).\mathsf{yes} \end{aligned}$$

and

$$m^{\ell}_{[\pi \mapsto \ell]}(\langle a_\pi \rangle x) = (a.(!\emptyset, a).x + b.(!\emptyset, b).\mathsf{yes}) \ \otimes\ (b.(!\emptyset, b).\mathsf{no} + a.(!\emptyset, a).\mathsf{yes}) \tag{4}$$

As the monitors in Example 4.10 indicate, a decentralized monitor synthesis for formulas of the form $\langle a_\pi \rangle \psi$ that is based on the encoding of (3) leads to monitors with a high degree of parallelism; for simplicity, the degree in Example 4.10 is reduced because we assumed to have just two actions. However, $|\mathsf{Act}| - 1$ parallel conjunctions are required in general. Alternatively, one could define a decentralized monitor synthesis directly for formulas of the form $\langle a_\pi \rangle \psi$ as follows:

$$m^{\ell}_{\sigma}(\langle a_\pi \rangle \psi) = \begin{cases} a.(!(\mathsf{rng}(\sigma) \backslash \{\ell\}), a).\mathrm{D}\mathsf{m}^{\ell}_{\sigma}(\psi) + \displaystyle\sum_{b \neq a} b.(!(\mathsf{rng}(\sigma) \backslash \{\ell\}), b).\mathsf{no} & \text{if } \sigma(\pi) = \ell \\ \displaystyle\sum_{b \in \mathsf{Act}} b.\Big((?\{\sigma(\pi)\}, a).\mathrm{D}\mathsf{m}^{\ell}_{\sigma}(\psi) \ + \displaystyle\sum_{b \neq a} (?\{\sigma(\pi)\}, b).\mathsf{no}\Big) & \text{otherwise} \end{cases}$$

This is essentially the synthesis for box formulas in Table 11 with no verdicts in place of yes. With this explicit rule for diamonds, (4) simply reduces to:

$$m^{\ell}_{[\pi\mapsto\ell]}(\langle a_\pi\rangle x) = a.(!\emptyset, a).x + b.(!\emptyset, b).\mathsf{no}$$

The synthesized monitor for diamond now contains no occurrence of any parallel operator.

5 Conclusion

We provided two methods to synthesize monitors for hyperproperties expressed as fragments of Hyper-recHML. Our first synthesis procedure constructs monitors that analyse hypertraces in a centralized manner and are guaranteed to correctly detect all violations of the respective formula, as long as it does not have a least fixed-point operator. Our second synthesis algorithm constructs monitors that operate in a decentralized manner and communicate with one another using multicast to share relevant information between them. The decentralized-monitor synthesis provides the same correctness guarantees as the centralized one, but is only defined for formulas with trace quantifiers that do not appear inside any fixed-point operator. This additional restriction, which is natural and present in many monitoring set-ups for hyperlogics, *e.g.* [10, 19, 23, 26, 30, 36], allows us to focus on examining the intricacies of monitoring in a decentralized setting with monitor communication. More precisely, it allows us to fix the σ in the synthesis function which, in turn, produces a *static* set of locations with which a monitor can communicate. Despite the restriction to PHyper-recHML, our synthesis algorithm still covers properties that were previously not even expressible, hence not monitorable, in state-of-the-art hyperlogics.

Of course, the picture is still incomplete: we have a centralized-monitor synthesis procedure for an expressive fragment of Hyper-recHML, whereas our decentralized-monitor synthesis deals with a more restricted fragment of that logic. It is not clear if this restriction is necessary; for example, a different decentralized-monitor synthesis for a larger fragment might be obtained by utilizing a different communication paradigm other than multicast, which was adopted in this study. In fact, we conjecture that broadcast communications might allow us to synthesize decentralized monitors for a larger Hyper-recHML fragment, including formulae that mix greatest fixed-points and quantifiers, like φ_a defined in (1); currently, monitors only send messages to the locations in the range of the specified σ. Another interesting direction is to allow monitors to infer information from communications they did not receive. A good starting point to explore such a synthesis algorithm (and prove its correctness) can be the synthesis properties in Definition 4.5. To fully delineate the power of decentralized monitoring, a maximality result in the spirit of those presented in [5, 7] is needed, which we intend to establish in the future.

Although we have focused on monitors that detect violations, we can also synthesize monitors that detect all satisfying hypertraces for the respective dual fragments of Hyper-recHML. Another direction we intend to pursue in future is the development of tools for monitoring Hyper-recHML specifications at runtime, based on the results of this article. We expect that our decentralised-monitor synthesis procedure can be implemented by generating a dedicated monitor for every location in a way that is very similar to the synthesis of μHML monitors presented in [3, 4, 11] and implemented in the tool `detectEr` available at `https://duncanatt.github.io/detecter/`.

Related Work. To the best of our knowledge, Agrawal and Bonakdarpour were the first to study RV for hyperproperties expressed in HyperLTL in [10], where they investigated monitorability for k-safety hyperproperties expressed in HyperLTL. They also gave a semantic characterization of monitorable k-safety hyperproperties, which is a natural extension to hyperproperties of the 'universal version' of the classic definition of monitorability presented by Pnueli-Zaks [7, 48]. In contrast to this work, we do not restrict ourselves to alternation-free formulas (see Eq. (1)) and every monitorable formula considered by Agrawal and Bonakdarpour can be expressed in our monitorable fragment. Brett et al. [23] improve on the work presented in [10] by presenting an algorithm for monitoring the full alternation-free fragment of HyperLTL. They also highlight challenges that arise when monitoring arbitrary HyperLTL formulas, namely (*i*) quantifier alternations, (*ii*) inter-trace dependencies and (*iii*) relative ordering of events across traces. Our decentralized-monitor synthesis addresses (*i*) by using the number of locations as an upper bound on the number of traces, and (*ii*) and (*iii*) via synchronized multicasts.

In [30], Finkbeiner et al. investigate RV for HyperLTL [26] formulas w.r.t. three different input classes, namely the bounded sequential, the unbounded sequential and the parallel classes. They also develop the monitoring tool RVHyper [29] based on the sequential algorithms developed for those input classes. The parallel class is closest to our set-up, since it consists in a *fixed* number of system executions that are processed synchronously.

Beutner et al. [15] study runtime monitoring for $\text{Hyper}^2\text{LTL}_{\text{fp}}$, a temporal logic that is interpreted over sets of *finite* traces of *equal length*. Unlike Hyper^2LTL [14], $\text{Hyper}^2\text{LTL}_{\text{fp}}$ permits quantification under temporal operators, which is also allowed in our logic Hyper-recHML. In contrast to HyperLTL, $\text{Hyper}^2\text{LTL}_{\text{fp}}$ features second-order quantification over sets of finite traces and can express properties like common knowledge.

In [36], Gustfeld et al. study automated analysis techniques for asynchronous hyperproperties and propose a novel automata-theoretic framework, the so-called alternating asynchronous parity automata, together with the fixed-point logic H_μ for expressing asynchronous hyperproperties. The logic H_μ has commonalities with PHyper-recHML, but it only allows for prenex formulas; moreover, its semantics progresses asynchronously on each trace. Properties such as "an atomic proposition does not occur at a certain level in the tree (of traces)" are not expressible in their logic H_μ, but can be described in Hyper-recHML.

Chalupa and Henzinger [25] explore the potential of monitoring for hyperproperties using prefix transducers. They develop a transducer language, called prefix expressions, give it an operational semantics over a hypertrace (reminiscent of the semantics in Section 4) and then implement it to assess the induced overheads. They show how transducers can use the writing capabilities as a method for monitor synchronization across traces, akin to the monitor communication and verdict aggregation of Section 4. Since transducers are, in principle, more powerful that passive monitors, additional guarantees are required to ensure that they do not interfere unnecessarily with system executions.

References

1 Luca Aceto, Antonis Achilleos, Elli Anastasiadi, and Adrian Francalanza. Monitoring hyperproperties with circuits. In Mohammad Reza Mousavi and Anna Philippou, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 42nd IFIP WG 6.1 International Conference, FORTE 2022*, volume 13273 of *LNCS*, pages 1–10. Springer, 2022.

2 Luca Aceto, Antonis Achilleos, Elli Anastasiadi, Adrian Francalanza, Daniele Gorla, and Jana Wagemaker. Centralized vs decentralized monitors for hyperproperties. *CoRR*, abs/2405.12882, 2024.

3 Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Léo Exibard, Adrian Francalanza, and Anna Ingólfsdóttir. A monitoring tool for linear-time μhml. In *COORDINATION*, volume 13271 of *LNCS*, pages 200–219. Springer, 2022.

4 Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Léo Exibard, Adrian Francalanza, and Anna Ingólfsdóttir. A monitoring tool for linear-time μHML. *Sci. Comput. Program.*, 232:103031, 2024.

5 Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Adventures in monitorability: from branching to linear time and back again. *Proc. ACM Program. Lang. POPL*, 3(52):1–29, 2019.

6 Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Testing equivalence vs. runtime monitoring. In *Models, Languages, and Tools for Concurrent and Distributed Programming*, volume 11665 of *LNCS*, pages 28–44. Springer, 2019.

7 Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. An operational guide to monitorability with applications to regular properties. *Softw. Syst. Model.*, 20(2):335–361, 2021.

8 Luca Aceto, Duncan Paul Attard, Adrian Francalanza, and Anna Ingólfsdóttir. Runtime Instrumentation for Reactive Components. In *ECOOP*, volume 313 of *LIPIcs*, pages 16:1–16:33. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

9 Luca Aceto, Ian Cassar, Adrian Francalanza, and Anna Ingólfsdóttir. On first-order runtime enforcement of branching-time properties. *Acta Informatica*, 60(4):385–451, 2023.

10 Shreya Agrawal and Borzoo Bonakdarpour. Runtime Verification of k-Safety Hyperproperties in HyperLTL. In *IEEE 29th Computer Security Foundations Symposium*, pages 239–252. IEEE Computer Society, 2016.

11 Duncan Paul Attard, Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Better late than never or: Verifying asynchronous components at runtime. In *FORTE*, volume 12719 of *LNCS*, pages 207–225. Springer, 2021.

12 Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. In Ezio Bartocci and Yliès Falcone, editors, *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *LNCS*, pages 1–33. Springer, 2018.

13 Raven Beutner and Bernd Finkbeiner. Software verification of hyperproperties beyond k-safety. In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022*, volume 13371 of *LNCS*, pages 341–362. Springer, 2022.

14 Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Niklas Metzger. Second-order hyperproperties. In *CAV (2)*, volume 13965 of *LNCS*, pages 309–332. Springer, 2023.

15 Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Niklas Metzger. Monitoring second-order hyperproperties. In Mehdi Dastani, Jaime Simão Sichman, Natasha Alechina, and Virginia Dignum, editors, *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024*, pages 180–188. ACM, 2024.

16 Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. Monitoring networks through multiparty session types. *Theor. Comput. Sci.*, 669:33–58, 2017.

17 Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida. A theory of design-by-contract for distributed multiparty interactions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory*, pages 162–176, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

18 Borzoo Bonakdarpour and Bernd Finkbeiner. Runtime verification for HyperLTL. In Yliès Falcone and César Sánchez, editors, *Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings*, volume 10012 of *LNCS*, pages 41–45. Springer, 2016.

19 Borzoo Bonakdarpour and Bernd Finkbeiner. The complexity of monitoring hyperproperties. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 162–174. IEEE Computer Society, 2018.

20 Borzoo Bonakdarpour, Pierre Fraigniaud, Sergio Rajsbaum, David A. Rosenblueth, and Corentin Travers. Decentralized asynchronous crash-resilient runtime verification. *J. ACM*, 69(5):34:1–34:31, 2022.

21 Borzoo Bonakdarpour, Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. Challenges in fault-tolerant distributed runtime verification. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications - 7th International Symposium, ISoLA 2016,*, volume 9953 of *LNCS*, pages 363–370, 2016.

22 Borzoo Bonakdarpour, César Sánchez, and Gerardo Schneider. Monitoring hyperproperties by combining static analysis and runtime verification. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Verification - 8th International Symposium, ISoLA 2018*, volume 11245 of *LNCS*, pages 8–27. Springer, 2018.

23 Noel Brett, Umair Siddique, and Borzoo Bonakdarpour. Rewriting-based runtime verification for alternation-free hyperltl. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 77–93, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

24 Ian Cassar, Adrian Francalanza, Claudio Antares Mezzina, and Emilio Tuosto. Reliability and fault-tolerance by choreographic design. In Adrian Francalanza and Gordon J. Pace, editors, *Proceedings Second International Workshop on Pre- and Post-Deployment Verification Techniques, PrePost@iFM 2017, Torino, Italy, 19 September 2017*, volume 254 of *EPTCS*, pages 69–80, 2017.

25 Marek Chalupa and Thomas A. Henzinger. Monitoring hyperproperties with prefix transducers. In *RV*, volume 14245 of *LNCS*, pages 168–190. Springer, 2023.

26 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust - Third International Conference, POST 2014*, volume 8414 of *LNCS*, pages 265–284. Springer, 2014.

27 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010.

28 E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic. volume 33, pages 151–178, 1986.

29 Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. RVHyper: A runtime verification tool for temporal hyperproperties. In *TACAS (2)*, volume 10806 of *LNCS*, pages 194–200. Springer, 2018.

30 Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. *Formal Methods Syst. Des.*, 54(3):336–363, 2019.

31 Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. A lower bound on the number of opinions needed for fault-tolerant decentralized run-time monitoring. *J. Appl. Comput. Topol.*, 4(1):141–179, 2020.

32 Adrian Francalanza. Consistently-detecting monitors. In *CONCUR*, volume 85 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

33 Adrian Francalanza. A Theory of Monitors. *Inf. Comput.*, 281:104704, 2021.

34 Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. Monitorability for the Hennessy-Milner logic with recursion. *Formal Methods Syst. Des.*, 51(1):87–116, 2017.

35 Adrian Francalanza, Andrew Gauci, and Gordon J. Pace. Distributed system contract monitoring. *J. Log. Algebraic Methods Program.*, 82(5-7):186–215, 2013.

36 Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Automata and fixpoints for asynchronous hyperproperties. *Proc. ACM Program. Lang.*, 5(POPL), jan 2021.

37 Christopher Hahn, Marvin Stenger, and Leander Tentrup. Constraint-based monitoring of hyperproperties. In Tomáš Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 115–131, Cham, 2019. Springer International Publishing.

38 Jun Inoue and Yoriyuki Yamagata. Operational semantics of process monitors. In *RV*, volume 10548 of *LNCS*, pages 403–409. Springer, 2017.

39 Limin Jia, Hannah Gommerstadt, and Frank Pfenning. Monitors and blame assignment for higher-order session types. In *POPL*, pages 582–594. ACM, 2016.

40 Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

41 Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, 2000.

42 Ruggero Lanotte, Massimo Merro, and Andrei Munteanu. A process calculus approach to detection and mitigation of PLC malware. *Theor. Comput. Sci.*, 890:125–146, 2021.

43 Ruggero Lanotte, Massimo Merro, and Andrei Munteanu. Industrial control systems security via runtime enforcement. *ACM Trans. Priv. Secur.*, 26(1):4:1–4:41, 2023.

44 Kim G. Larsen. Proof systems for satisfiability in Hennessy-Milner logic with recursion. *Theoretical Computer Science*, 72(2):265–288, 1990.

45 Claudio Antares Mezzina and Jorge A. Pérez. Causally consistent reversible choreographies: A monitors-as-memories approach. In *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming*, PPDP '17, page 127–138, New York, NY, USA, 2017. Association for Computing Machinery.

46 Iain Phillips. Refusal testing. *Theoretical Computer Science*, 50:241–284, 1987.

47 Amir Pnueli. The temporal logic of programs. In *FOCS'77, 18th IEEE Annual Symposium on Foundations of Computer Science, Proceedings*, pages 46–57. IEEE, 1977.

48 Amir Pnueli and Aleksandr Zaks. PSL model checking and run-time verification via testers. In *FM*, volume 4085 of *LNCS*, pages 573–586. Springer, 2006.

49 George M. Reed and A. W. Roscoe. The timed failures-stability model for CSP. *heoretical Computer Science*, 211(1–2):85–127, 1999.

50 A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall PTR, USA, 1997.

51 Moshe Y. Vardi. A temporal fixpoint calculus. In Jeanne Ferrante and Peter Mager, editors, *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 250–259. ACM Press, 1988.

52 Pierre Wolper. Temporal logic can be more expressive. *Inf. Control.*, 56(1/2):72–99, 1983.