# Monitoring Second-Order Hyperproperties

Raven Beutner
CISPA Helmholtz Center for Information Security
Germany

Bernd Finkbeiner
CISPA Helmholtz Center for Information Security
Germany

Hadar Frenkel
CISPA Helmholtz Center for Information Security
Germany

Niklas Metzger
CISPA Helmholtz Center for Information Security
Germany

## ABSTRACT

Hyperproperties express the relationship between multiple executions of a system. This is needed in many AI-related fields, such as knowledge representation and planning, to capture system properties related to knowledge, information flow, and privacy. In this paper, we study the monitoring of complex hyperproperties at runtime. Previous work in this area has either focused on the simpler problem of monitoring trace properties (which are sets of traces, while hyperproperties are sets of sets of traces) or on monitoring first-order hyperproperties, which are expressible in temporal logics with first-order quantification over traces, such as HyperLTL. We present the first monitoring algorithm for the much more expressive class of second-order hyperproperties. Second-order hyperproperties include system properties like common knowledge, which cannot be expressed in first-order logics like HyperLTL.

We introduce Hyper$^2$LTL$_f$, a temporal logic over finite traces that allows for second-order quantification over sets of traces. We study the monitoring problem in two fundamental execution models: (1) the parallel model, where a fixed number of traces is monitored in parallel, and (2) the sequential model, where an unbounded number of traces is observed sequentially, one trace after the other. For the parallel model, we show that the monitoring of the second-order hyperproperties of Hyper$^2$LTL$_f$ can be reduced to monitoring first-order hyperproperties. For the sequential model, we present a monitoring algorithm that handles second-order quantification efficiently, exploiting optimizations based on the monotonicity of subformulas, graph-based storing of executions, and fixpoint hashing. We present experimental results from a range of benchmarks, including examples from common knowledge and planning.

## KEYWORDS

Runtime Verification, Common Knowledge, Multi-agent Systems

**ACM Reference Format:**
Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Niklas Metzger. 2024. Monitoring Second-Order Hyperproperties. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.



*Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand.* © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

## 1 INTRODUCTION

Monitoring is a practical and scalable method for ensuring that the behavior of a complex system satisfies its formal specification. Unlike traditional verification techniques, such as model checking and theorem proving, monitoring does not analyze a system model, but instead works directly with the traces of the running system. Monitoring has been well studied for various types of trace properties, such as invariants or other properties expressible in standard temporal logics like linear-time temporal logic (LTL) [3, 15, 24]. By contrast, the study of monitoring algorithms for hyperproperties is still in an early stage.

Hyperproperties express relationships between multiple traces. Many properties related to knowledge, information flow, and privacy are hyperproperties and can, therefore, not be analyzed with methods for trace properties. The conceptual challenge is that while trace properties are sets of traces, hyperproperties are *sets of* sets of traces. So while the monitor of a trace property simply decides whether the observed trace is an element of the given set, the monitor of a hyperproperty must consider the entire set of traces produced by the system under observation (including any traces that will only be produced in the future) and decide whether this full set of traces is an element of the specified set of sets of traces.

Previous work on monitoring hyperproperties [1, 7, 8, 14] has focused on the special case of first-order hyperproperties, in particular on properties that can be expressed in the temporal logic HyperLTL [10]. HyperLTL extends LTL with first-order quantification over traces. First-order hyperproperties include many information-flow policies, such as noninterference and some simple notions of knowledge. Consider, for example, the property that an agent $i$ in a multi-agent system (MAS) knows that an LTL formula $\varphi$ is true whenever formula $\varphi$ is actually true. *Knowing* that $\varphi$ holds on a trace $\pi$ means that $\varphi$ must hold on all traces $\pi'$ that are indistinguishable from $\pi$ for agent $i$. The property is, therefore, expressed by the HyperLTL formula

$$\forall \pi. \forall \pi'. \, \varphi[\pi] \rightarrow (\pi \sim_i \pi' \rightarrow \varphi[\pi']),$$

where $\pi \sim_i \pi'$ denotes that agent $i$ cannot distinguish between $\pi$ and $\pi'$, and $\varphi[\pi]$ encodes that $\varphi$ holds on trace $\pi$. A HyperLTL monitor for this formula detects a violation as soon as a trace $\pi'$ is observed that is indistinguishable from some previously seen trace $\pi$ and yet $\pi'$ does not satisfy $\varphi$.

Many hyperproperties of interest, however, cannot be expressed with first-order quantifiers alone. A prominent example is common

knowledge [11, 23, 32]. *Common knowledge* (CK) requires that every agent $i$ not only knows that $\varphi$ is true, but, additionally, knows that every other agent knows that agent $i$ knows that $\varphi$ is true. The fact that every agent knows this, must, in turn, be known by every other agent, and so on. Formally, common knowledge refers to a set of traces that is closed under the indistinguishability relations of all agents, and requires that $\varphi$ holds on all traces in this set. Hyper$^2$LTL [6] extends HyperLTL with second-order quantification over sets of traces and can thus express properties like CK. Since model-checking Hyper$^2$LTL formulas is undecidable, the verification method for Hyper$^2$LTL is based on approximation techniques. The question arises whether or not monitoring second-order hyperproperties suffers from a similar limitation. In this paper, we show that while the precise answer depends on the underlying execution model, for a large class of second-order hyperproperties the monitoring problem can, in fact, be solved effectively.

As a specification logic for monitoring second-order hyperproperties in MASs, we introduce the temporal logic Hyper$^2$LTL$_f$. The main difference to Hyper$^2$LTL is that monitoring deals with *finite* rather than infinite traces; Hyper$^2$LTL$_f$ therefore has a finite-trace semantics. Hyper$^2$LTL$_f$ furthermore includes past-time temporal operators and allows for arbitrary nestings of temporal operators and first/second-order quantifiers. Common knowledge in a MAS with agents $1, \dots, n$ can be expressed as the Hyper$^2$LTL$_f$ formula:

$$\forall \pi.\, \varphi[\pi] \rightarrow \exists X.\, \pi \in X \wedge \Big(\forall \pi_1 \in X. \forall \pi_2.\, \big(\bigvee_{i=1}^{n} \pi_1 \sim_i \pi_2\big) \rightarrow \pi_2 \in X\Big) \wedge \forall \pi' \in X.\, \varphi[\pi'].$$

The second-order quantifier $\exists X$ specifies the existence of a set $X$ of traces so that $\pi$ is in $X$, and all traces that are indistinguishable from some trace in $X$ for some agent are also in $X$. Given this set, a monitor detects a violation by finding a trace in the set that does not satisfy $\varphi$.

Two fundamental execution models in the monitoring of hyperproperties are the parallel and sequential models (cf. [14]): in the *parallel model*, a fixed number of traces is monitored in parallel; in the *sequential model*, an unbounded number of traces is observed sequentially, one trace after the other.

For the parallel model, we show that the monitoring of the second-order hyperproperties expressed in Hyper$^2$LTL$_f$ can be reduced to monitoring first-order hyperproperties. For the sequential model, we show that the monitoring problem is undecidable in general, but becomes feasible for the practically relevant class of monotone second-order hyperproperties. A second-order hyperproperty is $\oplus$-monotone if its satisfaction on some set of traces implies that it is also satisfied on any superset of this set. Conversely, a second-order hyperproperty is $\ominus$-monotone if a violation on some set implies its violation on any superset. Monotonicity thus allows the monitor to provide *definitive* results that hold irrespectively of the traces that may still arrive in the future.

We introduce an inference system for monotonicity and present a monitoring algorithm that iteratively checks the given set of traces until it can produce a decisive answer. The algorithm has been implemented in a tool called MoSo. We report on encouraging evaluation results for MoSo on several benchmarks, including examples from common knowledge and planning.

*Related Work.* Giacomo and Vardi [19] first introduced a logic for interpreting LTL on finite traces, called LTL$_f$, specifically designed for AI systems. Since then, LTL$_f$ and its variants have been used, e.g., for model checking [2, 30], satisfiability analysis [16], synthesis [20, 21], and planning [9]. Knowledge in combination with LTL dates back to Fagin et al. [11] and has has found many applications in MASs [12, 25, 26]. Logics for hyperproperties have mostly been obtained by extending temporal logics with explicit quantification over traces or paths, such as HyperLTL [10], HyperQPTL [29], HyperPDL [22], HyperATL$^*$ [4, 5], and HyperLDL$_f$ [18]. Hyper$^2$LTL adds second-order quantification over *sets* of traces [6]. Tools for model-checking knowledge in multi-agent systems include MCK [17] and MCMAS [27], the latter was also extended to finite trace semantics [26]. While these approaches and tools implement solutions explicitly for knowledge, Hyper$^2$LTL$_f$ generalizes to more general second-order hyperproperties.

## 2 Hyper$^2$LTL$_f$

We consider a finite set of atomic propositions $AP$ and define $\Sigma := 2^{AP}$. We define Hyper$^2$LTL$_f$ as a finite-trace extension of Hyper$^2$LTL [6]. Let $\mathcal{V} = \{\pi, \pi', \pi_1, \dots\}$ be a set of (first-order) trace variables and $\mathcal{W} = \{X, Y, \dots\}$ a set of (second-order) set variables. We assume a special second-order variable $sys \in \mathcal{W}$ that we use to refer to the set of all system traces. Hyper$^2$LTL$_f$ formulas are defined by the following grammar:

$$\varphi := a_\pi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \ominus \varphi \mid \varphi\, \mathcal{U}\, \varphi \mid \varphi\, \mathcal{S}\, \varphi \mid \mathbb{Q}\pi \in X.\, \varphi \mid \mathbb{Q}X.\, \varphi$$

where $a \in \mathrm{AP}$, $\pi \in \mathcal{V}$, $X \in \mathcal{W}$, and $\mathbb{Q} \in \{\forall, \exists\}$ is a quantifier. In Hyper$^2$LTL$_f$, we have the future temporal operators (strong) *next* $\bigcirc$ and *until* $\mathcal{U}$, as well as their past counterparts *previously* $\ominus$ and *since* $\mathcal{S}$. We use the derived Boolean constants *true*, *false*, and connectives $\vee, \rightarrow, \leftrightarrow$, and the temporal operators *eventually* $\Diamond \varphi = true\, \mathcal{U}\, \varphi$, *once* $\boxminus \varphi = true\, \mathcal{S}\, \varphi$, *globally* $\Box \varphi = \neg \Diamond \neg \varphi$, and *historically* $\boxminus \varphi = \neg \boxminus \neg \varphi$.

*Semantics.* The semantics of Hyper$^2$LTL$_f$ is defined with respect to a trace length $m \in \mathbb{N}$ and a set of traces $\mathbb{T} \subseteq \Sigma^m$.[1] As for HyperLTL, we use a trace assignment $\Pi : \mathcal{V} \rightharpoonup \Sigma^m$ mapping trace variables in $\mathcal{V}$ to finite traces of length $m$. Additionally, we maintain a second-order assignment $\Delta : \mathcal{W} \rightharpoonup 2^{\Sigma^m}$ mapping variables to sets of finite traces of length $m$. The semantics is then as follows:

$\Pi, \Delta, i \vDash_{\mathbb{T}} a_\pi$ iff $a \in \Pi(\pi)(i)$

$\Pi, \Delta, i \vDash_{\mathbb{T}} \neg\varphi$ iff $\Pi, \Delta, i \nvDash_{\mathbb{T}} \varphi$

$\Pi, \Delta, i \vDash_{\mathbb{T}} \varphi_1 \wedge \varphi_2$ iff $\Pi, \Delta, i \vDash_{\mathbb{T}} \varphi_1$ and $\Pi, \Delta, i \vDash_{\mathbb{T}} \varphi_2$

$\Pi, \Delta, i \vDash_{\mathbb{T}} \bigcirc \varphi$ iff $i < m - 1$ and $\Pi, \Delta, i + 1 \vDash_{\mathbb{T}} \varphi$

$\Pi, \Delta, i \vDash_{\mathbb{T}} \ominus \varphi$ iff $i > 0$ and $\Pi, \Delta, i - 1 \vDash_{\mathbb{T}} \varphi$

$\Pi, \Delta, i \vDash_{\mathbb{T}} \varphi_1\, \mathcal{U}\, \varphi_2$ iff $\exists i \leq j < m.\, \Pi, \Delta, j \vDash_{\mathbb{T}} \varphi_2$ and $\forall i \leq k < j.\, \Pi, \Delta, k \vDash_{\mathbb{T}} \varphi_1$

$\Pi, \Delta, i \vDash_{\mathbb{T}} \varphi_1\, \mathcal{S}\, \varphi_2$ iff $\exists j \leq i.\, \Pi, \Delta, j \vDash_{\mathbb{T}} \varphi_2$ and $\forall i \geq k > j.\, \Pi, \Delta, k \vDash_{\mathbb{T}} \varphi_1$

[1] We restrict to traces of the same length to obtain simpler semantics. Without this restriction, we would have to deal with combinations of traces with different lengths, making traversal difficult. In practice, we can either pad traces to the length of the longest trace or crop them to the length of the shortest trace (cf. [13, 14]).

$$\Pi, \Delta, i \vDash_{\mathbb{T}} \mathbb{Q}\pi \in X.\, \varphi \quad \text{iff} \quad \mathbb{Q}t \in \Delta(X).\, \Pi[\pi \mapsto t], \Delta, i \vDash_{\mathbb{T}} \varphi$$
$$\Pi, \Delta, i \vDash_{\mathbb{T}} \mathbb{Q}X.\, \varphi \quad \text{iff} \quad \mathbb{Q}A \subseteq \mathbb{T}.\, \Pi, \Delta[X \mapsto A], i \vDash_{\mathbb{T}} \varphi$$

The temporal and boolean operators are evaluated as expected. Atomic formula $a_\pi$ holds in step $i$ if $a$ holds in the $i$the step on the trace bound to $\pi$. When quantifying over a trace $\pi \in X$, we quantify over all traces in its current second-order model (as given by $\Delta$), and when quantifying over a set of traces, we consider all possible subsets of $\mathbb{T}$. A set of traces $\mathbb{T} \subseteq \Sigma^m$ satisfies $\varphi$, written $\mathbb{T} \vDash \varphi$, if $\emptyset, [sys \mapsto \mathbb{T}], 0 \vDash_{\mathbb{T}} \varphi$. Note that we bind the second-order variable $sys$ to $\mathbb{T}$; by writing $\forall \pi \in sys$ and $\exists \pi \in sys$ we can thus quantify over the traces in $\mathbb{T}$ (as in HyperLTL).

*HyperLTL and Hyper$^2$LTL$_f$.* There are four key differences between the logics Hyper$^2$LTL$_f$ and HyperLTL: Firstly, Hyper$^2$LTL$_f$ adds past operators, which allow for exponentially more succinct definitions of properties [28] and offers a convenient syntax to specify knowledge properties (cf. Example 1). Secondly, the semantics of Hyper$^2$LTL$_f$ is defined w.r.t. finite traces, which occur much more frequently in MAS-related domains [2, 11, 19, 20]. Thirdly, Hyper$^2$LTL$_f$ allows quantification under temporal operators. In HyperLTL [10] or Hyper$^2$LTL [6], a formula consists of a quantifier prefix followed by a quantifier-free LTL body. In contrast, in Hyper$^2$LTL$_f$, we can interleave temporal operators and quantification. For example, $\forall \pi \in sys. \Diamond \forall \pi' \in sys. \Box(a_\pi \leftrightarrow a_{\pi'})$ states that on any trace, we can find a timepoint from which point on all other traces agree in $a$. Note that despite interleaving temporal operators and quantification, Hyper$^2$LTL$_f$ has a *linear-time semantics* (different from, e.g., HyperCTL$^*$ [10]), which is crucial in our monitoring setting. And lastly, and most importantly, Hyper$^2$LTL$_f$ features second-order quantification over *sets of traces*. Note that while second-order quantification in Hyper$^2$LTL [6] ranges over arbitrary sets of traces, Hyper$^2$LTL$_f$ quantifies over subsets of $\mathbb{T}$.

Example 1 (Eventual Knowledge). *As a running example, we use the sender-receiver MAS modeled by transition system in Figure 1, which is a variation of an example presented in [32]. In this system, agent 1 attempts to send a message until it is either received by agent 2, or delayed for one step and received afterward. The two agents each only have a partial view of the system: agent 1 observes the sending state (i.e., only observes AP $s$) and thus cannot distinguish if the message is immediately received or delayed for one step. Likewise, agent 2 only observes if the message was received (modeled by AP $r$). We want to express that whenever a message is received twice (i.e., two consecutive $r$s occur), agent 1 eventually knows that it was received. As in epistemic logics, an agent knows a property on a trace if it holds on all indistinguishable traces, which we can express as follows:*

$$\forall \pi \in sys. \big(\Diamond(r_\pi \wedge \bigcirc r_\pi)\big) \rightarrow \Diamond(\forall \pi' \in sys.\, \pi \sim_1 \pi' \rightarrow \Diamond r_{\pi'}),$$

*where we define $\pi \sim_1 \pi' := \boxminus(s_\pi \leftrightarrow s_{\pi'})$ to state that agent 1 cannot distinguish $\pi$ and $\pi'$ up to the current time step. Here, we quantify over all traces $\pi$ that visit the receiving state twice. For any such $\pi$, we require that* eventually, *all traces $\pi'$ that are – up to the current time point – indistinguishable from $\pi$, satisfy $\Diamond r_{\pi'}$. This property holds on the system in Figure 1. For example, consider the trace $s^n rr$, where sending takes $n$ steps and immediately afterward the message is received. In the last time step, this trace is indistinguishable (for agent 1) from $s^n rr$ and $s^n dr$, and both of these traces satisfy $\Diamond r$.* △

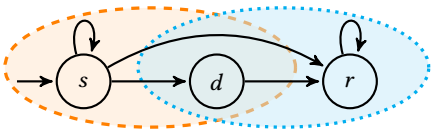


**Figure 1: A transition system modelling a sender-receiver MAS. Agent 1 only observes AP $s$, and thus cannot distinguish between the states in the blue (dotted) are. Agent 2 only observes AP $r$ and thus cannot distinguish between the states in the orange (dashed) area.**

*Fixpoint Formulas.* As already observed in [6], full second-order quantification is hard to handle algorithmically, particularly in a monitoring setting: If the monitor has seen $n$ finite traces, any of the $2^n$ subsets might qualify as a witness and need to be checked. Fortunately, full second-order quantification is not needed for most properties of interest. Instead, most second-order sets are defined in terms of a monotone fixpoint. Similar to [6], we extend the syntax of Hyper$^2$LTL$_f$ with a dedicated fixpoint construct, which allows for a convenient definition of such fixpoint-based sets of traces. Moreover, our monitoring algorithm can exploit the monotonicity properties of fixpoints. The extended syntax of Hyper$^2$LTL$_f$ is then the following:

$$\varphi := a_\pi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \ominus\varphi \mid \varphi\, \mathcal{U}\, \varphi \mid \varphi\, \mathcal{S}\, \varphi$$
$$\mid \mathbb{Q}\pi \in X.\, \varphi \mid \mathbb{Q}X.\, \varphi \mid \mathit{fix}(X, \xi_1, \ldots, \xi_k).\, \varphi.$$

The fixpoint operator $\mathit{fix}(X, \xi_1, \ldots, \xi_k).\, \varphi$ constructs a unique set of traces $X \in \mathcal{W}$ that can be used in $\varphi$. This set $X$ is uniquely defined by the fixpoint constraints $\xi_1, \ldots, \xi_k$ of the form

$$\forall \pi_1 \in X_1 \ldots \forall \pi_n \in X_n.\, \varphi_{step} \rightarrow \pi \in X, \tag{1}$$

where $X_1, \ldots, X_n \in \mathcal{W}$, $\pi, \pi_1, \ldots, \pi_n \in \mathcal{V}$, and $\varphi_{step}$ is a quantifier-free formula. To be well-formed, (1) all trace variables used in $\varphi_{step}$ must either be quantified outside or be one of $\pi_1, \ldots, \pi_n$; (2) all second-order variables $X_1, \ldots, X_n$ must be quantified outside or be equal to $X$, i.e., within the definition of $X$ we can quantify over traces in $X$ (as is usual for fixpoint definitions); and (3) $\pi$ must be quantified outside or be one of $\pi_1, \ldots, \pi_n$. Intuitively, Equation (1) states a requirement on traces that should be included in $X$. If we find traces $t_1 \in X_1, \ldots, t_n \in X_n$ that, together with the sets and traces quantified outside of $\mathit{fix}(X, \xi_1, \ldots, \xi_k).\, \varphi$, satisfy $\varphi_{step}$, then trace $\pi$ should be added to $X$. In our semantics, we define $\Pi, \Delta, i \vDash_{\mathbb{T}} \mathit{fix}(X, \xi_1, \ldots, \xi_k).\, \varphi$ iff

$$\Pi, \Delta\big[X \mapsto \mathit{sol}(\Pi, \Delta, i, X, \xi_1, \ldots, \xi_k)\big], i \vDash_{\mathbb{T}} \varphi,$$

where $\mathit{sol}(\Pi, \Delta, i, X, \xi_1, \ldots, \xi_k)$ denotes the unique solution to the fixpoint definition of $X$. Formally, $\mathit{sol}(\Pi, \Delta, i, X, \xi_1, \ldots, \xi_k)$ is the *smallest* set of traces such that *for all* $\xi_i$ we have

$$\Pi, \Delta[X \mapsto \mathit{sol}(\Pi, \Delta, i, X, \xi_1, \ldots, \xi_k)], i \vDash_{\mathbb{T}} \xi_i.$$

Note that the fixpoint solution $\mathit{sol}(\Pi, \Delta, i, X, \xi_1, \ldots, \xi_k)$ is uniquely defined.[2] Using this fixpoint construct, we can easily express CK:

[2] Given $\Pi, \Delta, i$, let $f : 2^{\mathbb{T}} \rightarrow 2^{\mathbb{T}}$ be the function that – given a current model for X – returns X with all traces that should be added according to one of the $\xi_i$. It is easy to see that $f$ is monotone in the subset order on $2^{\mathbb{T}}$. Each fixpoint of $f$ now corresponds to a set $X$ that satisfies $\xi_1, \ldots, \xi_k$. By Knaster-Tarski [31], there exists a *unique* smallest fixpoint of $f$, which is exactly $\mathit{sol}(\Pi, \Delta, i, X, \xi_1, \ldots, \xi_k)$.

Example 2 (Common Knowledge and Fixpoints). *We illustrate our fixpoint construct by continuing on Example 1. Assume we do not want to state that on all traces where $\Diamond(r \wedge \bigcirc r)$ holds, agent 1 knows that $\Diamond r$ holds (cf. Example 1), but rather that it is* common knowledge *in the group of agents 1 and 2 that $\Diamond r$ holds. As argued in Section 1, this requires us to iteratively compute the set of all traces that cannot be distinguished by some agent. While the formulation of CK in Section 1 expressed this using full second-order quantification, we can observe that the set we are interested in is actually defined by a fixpoint:*

$$\forall \pi \in sys.\big(\Diamond(r_\pi \wedge \bigcirc r_\pi)\big) \rightarrow \Diamond\Big(fix(X, \xi_1, \xi_2).\, \forall \pi' \in X.\, \Diamond r_{\pi'}\Big),$$

*where $\xi_1 := true \rightarrow \pi \in X$, and $\xi_2$ is defined as*

$$\forall \pi_1 \in X. \forall \pi_2 \in sys.\, \big(\pi_1 \sim_1 \pi_2 \vee \pi_1 \sim_2 \pi_2\big) \rightarrow \pi_2 \in X.$$

*Here, we define $\pi \sim_2 \pi' := \boxminus(r_\pi \leftrightarrow r_{\pi'})$, similar to $\sim_1$ in Example 1. For each $\pi$, we specify a fixpoint-defined set $X$. This set $X$ includes $\pi$ (constraint $\xi_1$), and whenever we find some $\pi_1 \in X$ that some agent cannot distinguish from some $\pi_2 \in sys$, we add $\pi_2$ to $X$ (constraint $\xi_2$). Then, we state that all traces $\pi'$ in $X$ should satisfy $\Diamond r_{\pi'}$. The above formula does not hold on the MAS modeled in Figure 1. Consider the trace $\pi = sr^n$. It is easy to see that*

$$sr^n \sim_1 sdr^{n-1} \sim_2 s^2r^{n-1} \sim_1 s^2dr^{n-2} \sim_2 \ldots \sim_2 s^nr \sim_1 s^nd.$$

*Consequently, the trace $s^nd$ will be included in $X$, disproving that $\Diamond r$ is common knowledge on $\pi$.* △

# 3 MONITORABILITY

We are interested in *monitoring* a Hyper$^2$LTL$_f$ formula, i.e., we do not have access to the system as a whole but rather observe executions of the system and conclude whether the set of traces we have seen so far suffices to conclude the satisfaction or violation of the property. This is, unsurprisingly, not possible for all properties. Some properties are not *monitorable*, i.e., never allow a positive or negative answer from the monitor.

## 3.1 Monitorability of Trace Properties

Let us recall the concept of monitorability in the simpler setting of a trace property, where we observe one step of the trace at a time.

Definition 1. *A trace property $L \subseteq \Sigma^*$ is monitorable if*

$$\forall u \in \Sigma^*. \exists v \in \Sigma^*. (\forall w \in \Sigma^*. uvw \in L) \vee (\forall w \in \Sigma^*. uvw \notin L).$$

Intuitively, the definition states that – whatever finite trace $u$ we start from – some extension $uv$ of $u$ allows the monitor to report the satisfaction or violation of $L$. This can either be the case because all extensions of $uv$ satisfy $L$ (the first disjunct) or all extensions violate $L$ (the second disjunct).[3] In this paper, we are not monitoring a trace property but a hyperproperty, i.e., we may need to observe multiple traces. Depending on the setting in which the monitor is deployed, there can be different modes in which those traces are presented to the monitor. We focus on the *parallel model* and the *sequential model* [14]. In the former, the number of traces is fixed *a priori*, and the monitor observes consecutive time steps one by one. In the latter model, the monitor observes the (finite) traces sequentially, increasing the cardinality of the trace set in every time step. We discuss both models and their impact on the monitorability of Hyper$^2$LTL$_f$ properties in the following.

[3]The sets $good(L) := \{u \in \Sigma^* \mid \forall v \in \Sigma^*. uv \in L\}$ and $bad(L) := \{u \in \Sigma^* \mid \forall v \in \Sigma^*. uv \notin L\}$ described by the disjunctions are commonly referred to as the set of good and bad prefixed, respectively.

## 3.2 The Parallel Model

In the parallel model, the number of traces (executions) is fixed to some number $b \in \mathbb{N}$, and each time step reveals an additional position on each of the traces. Similar to the monitorability of trace properties, the monitor thus gets presented more and more positions on the same $b$ traces. We refer to [14] for more details.

As the parallel model *fixes* the number of sessions (traces), it turns out that second-order quantification does not yield any additional expressiveness. Intuitively, quantifying over a set of traces is equivalent to quantifying over some finite subset of at most $b$ traces, which is expressible in HyperLTL with first-order trace quantification. Formally, given bound $b \in \mathbb{N}$ and a Hyper$^2$LTL$_f$ formula $\varphi$, we can unfold $\varphi$ into a formula that uses no second-order quantification and is equivalent on any set of traces with at most $b$ traces. We maintain a partial function $M : \mathcal{W} \rightharpoonup 2^{\mathcal{V}}$ mapping second-order variables to a set of trace variables. We then recursively define the Hyper$^2$LTL$_f$ formula $[\![\varphi]\!]_{b,M}$ as follows:

$$\begin{aligned}
[\![a_\pi]\!]_{b,M} &:= a_\pi \\
[\![\circ\varphi]\!]_{b,M} &:= \circ[\![\varphi]\!]_{b,M} \quad \text{for } \circ \in \{\neg, \bigcirc, \ominus\} \\
[\![\varphi_1 \circ \varphi_2]\!]_{b,M} &:= [\![\varphi_1]\!]_{b,M} \circ [\![\varphi_2]\!]_{b,M} \quad \text{for } \circ \in \{\wedge, \mathcal{U}, \mathcal{S}\} \\
[\![\exists X.\, \varphi]\!]_{b,M} &:= \exists \pi_1, \ldots, \pi_b.\, [\![\varphi]\!]_{b,M[X \mapsto \{\pi_1,\ldots,\pi_b\}]} \\
[\![\forall X.\, \varphi]\!]_{b,M} &:= \forall \pi_1, \ldots, \pi_b.\, [\![\varphi]\!]_{b,M[X \mapsto \{\pi_1,\ldots,\pi_b\}]} \\
[\![\exists \pi \in X.\, \varphi]\!]_{b,M} &:= \bigvee_{\pi' \in M(X)} [\![\varphi]\!]_{b,M}[\pi'/\pi] \\
[\![\forall \pi \in X.\, \varphi]\!]_{b,M} &:= \bigwedge_{\pi' \in M(X)} [\![\varphi]\!]_{b,M}[\pi'/\pi]
\end{aligned}$$

Here, $\varphi[\pi'/\pi]$ denotes the formula in which all free occurrences of $\pi$ are replaced with $\pi'$. For quantifier-free formulas, we simply maintain the structure. Since we assume that there are at most $b$ traces, we can replace second-order quantification over a set $X$ with first-order quantification over $b$ *fresh* (first-order) trace variables $\pi_1, \ldots, \pi_b$. We record the first-order trace variables that we used to replace $X$ within the auxiliary mapping $M$. For first-order quantification $\forall \pi \in X$ (resp. $\exists \pi \in X$), we then conjunctively (resp. disjunctively) consider all traces $\pi' \in M(X)$ in place of $\pi$. A simple induction shows:

Proposition 1. *Let $\varphi$ be any Hyper$^2$LTL$_f$ formula and $b \in \mathbb{N}$. For any set of trace $\mathbb{T}$ with $|\mathbb{T}| \leq b$ we have $\mathbb{T} \vDash \varphi$ iff $\mathbb{T} \vDash [\![\varphi]\!]_{b,\emptyset}$.*

Assuming a fixed bound $b \in \mathbb{N}$ (as in the parallel model), Proposition 1 states that monitoring a Hyper$^2$LTL$_f$ formula $\varphi$ is equivalent to monitoring $[\![\varphi]\!]_{b,\emptyset}$. As $[\![\varphi]\!]_{b,\emptyset}$ uses *no* second-order quantification, we can apply all the techniques and tools developed for first-order logics like HyperLTL [14].

## 3.3 The Sequential Model

We now consider the sequential model, the main object of study in this paper. In this model, traces arrive one at a time and the number of traces is *unbounded*. Our model is similar to the sequential

model used in [14] for HyperLTL, with the exception that traces in our model are of finite (instead of infinite) length. This focus on finite traces makes our approach (and tool) applicable to many MAS-related settings, where executions are finite, and tasks are subsequent system executions.

Definition 2. *A hyperproperty $H \subseteq 2^{\Sigma^*}$ is monitorable in the sequential model if*

$$\forall U \subseteq \Sigma^*.\, \exists V \subseteq \Sigma^*.\, \big(\forall W \subseteq \Sigma^*.\, (U \cup V \cup W) \in H\big) \vee \big(\forall W \subseteq \Sigma^*.\, (U \cup V \cup W) \notin H\big).$$

That is, for any $U$ there exists some $V$ such that $U \cup V$ allows some definitive answer by the monitor. This can either be the case because all extensions of $U \cup V$ satisfy $H$ (the first disjunct) or all extensions violate $H$ (the second disjunct).

Example 3. *Consider the CK formula in Example 2. The formula is monitorable in the sequential model: We can always add traces to ensure that CK does not hold, i.e., add some indistinguishable trace that does not satisfy $\Diamond r$. No matter what additional traces are observed, CK remains violated.* △

Theorem 1. *Deciding if a Hyper$^2$LTL$_f$ formula is monitorable in the sequential model is undecidable.*

Proof. Finkbeiner et al. [14] showed that monitorability is undecidable for HyperLTL with finite traces in the sequential model. Since HyperLTL with finite-trace semantics is a strict subset of Hyper$^2$LTL$_f$, monitorability of Hyper$^2$LTL$_f$ is undecidable. □

## 4 MONOTONICITY

In the following, we focus on the sequential model. Theorem 1 rules out the possibility of an algorithm that monitors all possible Hyper$^2$LTL$_f$ formulas. Instead, we focus on fragments of formulas for which we can provide definitive answers. Given a finite set of traces, we want to provide a monitoring result irrespective of what traces will arrive in the future. To ensure this, the truth value of the formula may not change when additional traces arrive; that is, it should be *monotone*. We distinguish between $\oplus$-monotonicty and $\ominus$-monotonicty. In $\oplus$-monotonicty, any model that *satsfies* the formula will continue to satisfy it no matter what traces are added. In $\ominus$-monotonicty, any model that *violates* the formula will continue to violate it no matter what traces are added.

Definition 3. *A Hyper$^2$LTL$_f$ formula $\varphi$ is $\oplus$-monotone (resp. $\ominus$-monotone) if for any set of traces $\mathbb{T}$ such that $\mathbb{T} \vDash \varphi$ (resp. $\mathbb{T} \nvDash \varphi$), for any larger set $\mathbb{T}' \supseteq \mathbb{T}$, we have $\mathbb{T}' \vDash \varphi$ (resp. $\mathbb{T}' \nvDash \varphi$).*

We say that a formula is monotone if it is either $\oplus$-monotone or $\ominus$-monotone. Once we detect that a $\oplus$-monotone (resp. $\ominus$-monotone) formula holds (resp. is violated) on the set of traces seen so far, our monitor can conclude that $\varphi$ holds (resp. does not hold) regardless of what traces will be observed in the future. The CK formula in Example 2 is $\ominus$-monotone, i.e., whenever the current set of traces violates CK, no additional set of traces can change it. In many cases, monotonicity implies monitorability:

Proposition 2. *Let $\varphi$ be a Hyper$^2$LTL$_f$ formula. If one of the following holds, then $\varphi$ is monitorable.*

$$\frac{}{\Gamma \vdash a_\pi : \oplus} \qquad \frac{}{\Gamma \vdash a_\pi : \ominus} \qquad \frac{\Gamma \vdash \varphi : \oplus}{\Gamma \vdash \neg\varphi : \ominus} \qquad \frac{\Gamma \vdash \varphi : \ominus}{\Gamma \vdash \neg\varphi : \oplus}$$

$$\frac{\circ \in \{\bigcirc, \ominus\} \qquad \Gamma \vdash \varphi : \alpha}{\Gamma \vdash \circ\, \varphi : \alpha} \qquad \frac{\circ \in \{\wedge, \mathcal{U}, \mathcal{S}\} \qquad \begin{array}{c}\Gamma \vdash \varphi_1 : \alpha \\ \Gamma \vdash \varphi_2 : \alpha\end{array}}{\Gamma \vdash \varphi_1 \circ \varphi_2 : \alpha}$$

$$\frac{X \in \Gamma \qquad \Gamma \vdash \varphi : \oplus}{\Gamma \vdash \exists \pi \in X.\, \varphi : \oplus} \qquad \frac{X \in \Gamma \qquad \Gamma \vdash \varphi : \ominus}{\Gamma \vdash \forall \pi \in X.\, \varphi : \ominus}$$

$$\frac{\Gamma \cup \{X\} \vdash \varphi : \alpha}{\Gamma \vdash \mathit{fix}(X, \xi_1, \ldots, \xi_k).\, \varphi : \alpha} \qquad \frac{\Gamma \vdash \varphi : \alpha}{\Gamma \vdash \mathbb{Q}X.\, \varphi : \alpha}$$

**Figure 2: Inference system for monotonicity**

*(1) $\varphi$ is $\oplus$-monotone and has at least one finite model (i.e., a finite set of traces $\mathbb{T}$ such that $\mathbb{T} \vDash \varphi$).*
*(2) $\varphi$ is $\ominus$-monotone and $\neg\varphi$ has at least one finite model.*

The proof directly follows from the definitions of monotonicity (Definition 3) and monitorability (Definition 2). The converse statement does not hold, even for the first-order fragment of Hyper$^2$LTL$_f$.

Example 4. *Consider the formula*

$$\varphi := \forall\pi.\exists\pi'.(\pi \neq \pi' \wedge \square(a_\pi \leftrightarrow a_{\pi'})) \vee \exists\pi''.\Diamond(b_{\pi''})$$

*and the trace sets $\mathbb{T} := \{\{a,c\}^m, \{a\}^m\}$, $\mathbb{T}' := \mathbb{T} \cup \{\{c\}^m\}$, and $\mathbb{T}'' := \mathbb{T}' \cup \{\{b\}^m\}$. It is easy to see that $\mathbb{T} \vDash \varphi$, $\mathbb{T}' \nvDash \varphi$, and $\mathbb{T}'' \vDash \varphi$. Formula $\varphi$ thus cannot be monotone. Yet, $\varphi$ is monitorable: for every set $U$, we can add $V := \{\{b\}^m\}$, and every extension of $U \cup V$ satisfies $\varphi$ (cf. Definition 2).* △

*An Inference System for Monotonicity.* In our monitoring algorithm, we use monotonicity to provide definitive monitoring answers. To statically determine the monotonicity of (sub)formulas, we use a deductive (type-like) inference system. The judgments in our system are of the form $\Gamma \vdash \varphi : \alpha$, where $\alpha \in \{\oplus, \ominus\}$, and $\Gamma = \{X_1, \ldots, X_n\}$ is a context assumption. Intuitively, $X \in \Gamma$ assumes that $X$ has a unique model that only grows when more traces arrive (i.e., the model is monotonically increasing); and $\Gamma \vdash \varphi : \oplus$ (resp. $\Gamma \vdash \varphi : \ominus$) implies that, under the context assumptions in $\Gamma$, $\varphi$ is $\oplus$-monotone (resp. $\ominus$-monotone). The inference rules of our system are depicted in Figure 2. Most rules in our system are straightforward: atomic propositions are both $\oplus$-monotone and $\ominus$-monotone, temporal operators preserve monotonicity, and negation "flips" monotonicity. More interesting are the rules for quantification. A formula $\varphi = \exists\pi \in X.\, \varphi'$ is $\oplus$-monotone if $\varphi'$ is $\oplus$-monotone and $X \in \Gamma$ (i.e., the model of $X$ only increases): assume that $\varphi$ holds on the current set of traces $\mathbb{T}$, i.e., there exists a trace $t \in X$ that is a witness for the satisfaction of $\varphi'$. As $X \in \Gamma$, the model of $X$ can only grow when more traces are added to $\mathbb{T}$, so we can always reuse $t$ as a witness. Likewise, models of a fixpoint-defined second-order variable $X$ only grow larger with the arrival of new traces, so we add $X$ to $\Gamma$. For full second-order quantification, we can make no assumption on how model(s) behave when future traces arrive, so we add no assumptions. In our inference system, we initially set the

**Algorithm 1** Monitoring algorithm for Hyper$^2$LTL$_f$ formula $\varphi$

```
1 monMap = computeMonotonicity(φ)
2 𝕋 = ∅
3 while (t = getNextTrace()):
4   𝕋 = 𝕋 ∪ {t}
5   res = check(∅,∅,0,𝕋,φ)
6   if res = true and ⊕ ∈ monMap(φ) then return SAT
7   if res = false and ⊖ ∈ monMap(φ) then return UNSAT
```

context to be $\Gamma = \{sys\}$, as the set of all traces (bound to the special variable $sys \in \mathcal{W}$) only grows larger. An easy induction shows:

Proposition 3. *Assume $\{sys\} \vdash \varphi : \oplus$ (resp. $\{sys\} \vdash \varphi : \ominus$), then $\varphi$ is $\oplus$-monotone (resp. $\ominus$-monotone).*

Note that our system is not complete, i.e., the converse of Proposition 3 does not hold. It is easy to see that our system allows us to conclude that the CK property in Example 2 is $\ominus$-monotone.

## 5 MONITORING ALGORITHM

We depict our basic monitor for a formula $\varphi$ in Algorithm 1. As a first step, we (1) compute a monotonicity map *monMap* using the inference system from Figure 2. Here, *monMap* maps each *subformula* $\varphi'$ of $\varphi$ to $monMap(\varphi') \in 2^{\{\oplus,\ominus\}}$ indicating whether $\varphi'$ – in a given context $\Pi, \Delta, i$ – is monotone; and (2) initialize the set $\mathbb{T}$ of system traces to the empty set. During the execution of the monitor, whenever a new trace $t$ arrives, we add it to $\mathbb{T}$ and check if $\mathbb{T}$ satisfies $\varphi$ by calling function `check` (which we discuss in Section 5.1). As we argued in Section 4, our monitor can only provide a definitive `SAT` answer in case $\varphi$ is $\oplus$-monotone and the current traces satisfy $\varphi$, and `UNSAT` in case it is $\ominus$-monotone and the current set of traces violates $\varphi$. Otherwise, we await further traces and repeat.

### 5.1 Incremental Model Checking

The core of our monitoring algorithm lies in our recursive model-checking function `check` given in Algorithm 2. On a high level, `check` casts the semantics of Hyper$^2$LTL$_f$ into an executable program. For now, we ignore the program fragments marked with a gray background; these concern the hashing-based optimizations we will discuss in Section 5.3. In line 6, we match on the structure of $\varphi$. We only include a selection of cases (the others can be inferred easily). In case $\varphi$ is an AP, we can immediately decide its truth value (line 7); for the cases of negation, conjunction, and temporal operators we perform the expected recursive call(s). For first-order quantification (line 9), we iteratively check all traces assigned by $\Delta$. For full second-order quantification (line 16), we check all subsets of $\mathbb{T}$ . In the case of fixpoint-based second-order quantification (line 21), we compute the (unique) solution to the fixpoint definition (line 23) by calling `computeFix` (discussed in Section 5.2).

### 5.2 Fixpoint Computation

As we argue in Section 2, fixpoints are expressive enough for most practical properties and are much easier to handle algorithmically. Given a set $\mathbb{T}$ with $n$ traces, we can compute the fixpoint in polynomial time solution using Knaster-Tarski fixpoint iteration [31] instead of trying all possible $2^n$ subsets of $\mathbb{T}$ (as needed for full second-order quantification). The input to Algorithm 3 is the fixpoint formula and the current set $A$ of traces in the (to-be) fixpoint set. Initially, $A = \emptyset$. We check if $A$ satisfies the fixpoint constraints $\xi_1, \ldots, \xi_k$: if we find traces $t_1 \in \Delta(X_1), \ldots, t_n \in \Delta(X_n)$ that satisfy the step constraint (which we check via a mutually recursive call in line 6), we add the trace required by the fixpoint constraint to $A$ and repeat via a recursive call (line 7).

**Algorithm 2** Incremental model-checking

```
1  def check(Π,Δ,i,𝕋,φ):
2    if ⊕ ∈ monMap(φ) and h_sat(Π,Δ,i,φ) = true then
3      return true
4    if ⊖ ∈ monMap(φ) and h_sat(Π,Δ,i,φ) = false then
5      return false
6    res = match φ with
7      | a_π: if a ∈ Π(π)(i) then return true else return false
8      | ¬φ': return (not (check(Π,Δ,i,𝕋,φ')))
9      | ∃π ∈ X. φ':
10       A = if h_wit(Π,Δ,i,φ) = t then order(t,Δ(X)) else Δ(X)
11       for t in A:
12         if check(Π[π ↦ t],Δ,i,𝕋,φ') then
13           h_wit(Π,Δ,i,φ) = t
14           return true
15       return false
16     | ∃X. φ':
17       for A ⊆ 𝕋:
18         if check(Π,Δ[X ↦ A],i,𝕋,φ') then
19           return true
20       return false
21     | fix(X,ξ_1,...,ξ_k). φ':
22       A' = if h_fix(Π,Δ,i,fix(X,ξ_1,...,ξ_k)) = A'' then A'' else ∅
23       A = computeFix(Π,Δ,i,𝕋,fix(X,ξ_1,...,ξ_k),A')
24       h_fix(Π,Δ,i,fix(X,ξ_1,...,ξ_k)) = A
25       return check(Π,Δ[X ↦ A],i,𝕋,φ')
26   h_sat(Π,Δ,i,φ) = res
27   return res
```

### 5.3 Monitoring Optimizations

When implemented without optimizations, Algorithm 2 checks the Hyper$^2$LTL$_f$ formula on the current set of traces, re-performing this check whenever a new trace arrives. This is often infeasible in practice, as with each new trace, the number of traces and, therefore, the verification time increases. Instead, whenever we add a trace $t$ to the current set of traces $\mathbb{T}$, we want to reuse as much of the computation we have already performed on $\mathbb{T}$ in previous steps. We discuss three areas where such re-usage is possible: hashing of verification results for subformulas, hashing of fixpoint results, and hashing of witnesses.

*SAT Hashing.* Our first optimization is the hashing of verification results of the `check` for *subformulas*. If – for a given evaluation context $(\Pi, \Delta, i)$ – a $\oplus$-monotone (resp. $\ominus$-monotone) subformula $\varphi$ was satisfied (resp. violated) in a previous iteration, then $\varphi$ remains satisfied (resp. violated) in context $(\Pi, \Delta, i)$, even if more traces are added to $\mathbb{T}$. We therefore use a (hashing) function $h_{sat}$ (initially set to the function with empty domain) and check if we have hashed $(\Pi, \Delta, i, \varphi)$ (Algorithm 2, lines 2 and 4). Here, we, e.g.,

**Algorithm 3** Fixpoint computation

```
1 def computeFix(Π, Δ, i, 𝕋, fix(X, ξ1, ..., ξk), A):
2   Δ' = Δ[X ↦ A]
3   for ∀π1 ∈ X1 ... ∀πn ∈ Xn. φstep → π ∈ X in {ξ1, ..., ξk}:
4     for t1 ∈ Δ'(X1), ..., tn ∈ Δ'(Xn):
5       Π' = Π[π1 ↦ t1, ..., πn ↦ tn]
6       if Π'(π) ∉ A and check(Π', Δ', i, 𝕋, φstep) then
7         return computeFix(Π, Δ, i, 𝕋, fix(X, ξ1, ..., ξk), A ∪ {Π'(π)})
8   return A
```

write $h_{sat}(\Pi, \Delta, i, \varphi)$ = true to denote that $(\Pi, \Delta, i, \varphi)$ is contained in the domain of $h_{sat}$ and maps to true. If this hashing does not suffice, we evaluate $\varphi$ and update $h_{sat}$ in line 26.

*Fixpoint Hashing.* The second optimization we use is hashing fixpoint solutions. In any given content $(\Pi, \Delta, i)$, the fixpoint solution only increases when new traces are added to $\mathbb{T}$. When we compute a fixpoint solution (Algorithm 2, line 23), we thus do not need to restart the computation from scratch. Instead, we hash the solution from the previous iteration using function $h_{fix}$. If we hashed $(\Pi, \Delta, i, fix(X, \xi_1, \dots, \xi_k))$, we can this previous solution as the starting point for the current fixpoint computation. Otherwise, we start the computation from the empty set (line 22). After calling computeFix, we update $h_{fix}$ and map $(\Pi, \Delta, i, fix(X, \xi_1, \dots, \xi_k))$ to the newly computed solution $A$ (line 24).

*Witness Hashing.* Our last hashing-based optimization concerns the analysis of first-order quantification. In line 11, we need to iterate over all traces in the current model. While we cannot avoid this in the worst case, we can optimize the order in which we explore the traces. Concretely, whenever we find a witness trace $t$, we store it using the $h_{wit}$ hashing function (line 13). In the next iteration, we attempt to reorder $\Delta(X)$ such that the witness from the previous iteration will be explored first (line 10). Intuitively, if $t$ is a witness for the satisfaction of some formula $\exists \pi.\varphi'$, then $t$ remains a promising witness when more traces are added to $\mathbb{T}$; exploring $t$ early can thus save significant time.

*Prefix and Postfix Trees.* Our last optimization is applicable to all hashing techniques above. Instead of storing the set $\mathbb{T}$ of all traces explicitly as a list, we store it as a tree, i.e., we store all traces based on their common prefixes (resp. postfixes). This has two advantages: It reduces memory overhead, particularly in cases where many traces share parts; and, secondly, a tree-based representation allows for even more efficient hashing. For example, if at time step $i$, we compute a fixpoint solution based only on future temporal operators (i.e., no past modalities), then all traces that, starting from step $i$, share the same postfix can be treated as equal during the fixpoint computation. We can thus lift the computation and operate on nodes in the postfix tree rather than on concrete traces.

# 6 EXPERIMENTS

We have implemented our monitoring algorithm and the optimizations from Section 5.3 in a tool called MoSo. In this section, we demonstrate that MoSo can monitor complex second-order hyperproperties that are out of reach of existing monitoring and model-checking frameworks.

**Table 1: We monitor CK in Example 2 for varying trace lengths and trace numbers (# traces). We report MoSo's average runtime in seconds ($t$).**

| **length** | 20 | 30 | 40 | 50 | 60 | 70 | 80 |
|---|---|---|---|---|---|---|---|
| **# traces** | 35 | 55 | 75 | 95 | 115 | 135 | 155 |
| $t$ | 0.51 | 1.51 | 5.98 | 18.40 | 48.83 | 111.52 | 230.71 |

**Table 2: We monitor CK in the muddy children's game. We report MoSo's average running time in seconds ($t$). We also depict the runtime on the non-fixpoint-based formulation of CK ($t_{noFix}$). The timeout (TO) is set to 1 min.**

| **# children** | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| $t$ | 0.1 | 0.2 | 0.2 | 0.2 | 0.3 | 0.4 | 0.4 | 0.4 |
| $t_{noFix}$ | 0.1 | 1.1 | TO | TO | TO | TO | TO | TO |

## 6.1 Running Example

The first experiment is the scalability of MoSo in the length of the traces. We verify the formula in Example 2 against traces of the running example in Figure 1. We generate traces for each instance of lengths 20, 30, 40, ..., 80 and measure the time it takes to find the violation for the starting trace $s^n r^n$ in Table 1. The monitor correctly concludes in all runs that the common knowledge formula is violated.

## 6.2 Common Knowledge in MASs

We monitor common knowledge in a scalable instance of the muddy children's puzzle [11]. The muddy children puzzle is a MAS between children $1, \dots, n$, where a number of rounds of communication are used to achieve common knowledge about how many children are muddy. For each child (agent) $i$, we have an AP $m_i$ indicating if $i$ is muddy and an AP $d_i$ indicating if $i$ declared that it is muddy. We can then express that the muddiness of all children is CK after $b \in \mathbb{N}$ steps of communication (which we call the *communication-bound*) as the following Hyper$^2$LTL$_f$ formula $\varphi_{mud}$:

$$\forall \pi \in sys. \bigcirc^b fix(X, \xi_1, \xi_2). \forall \pi_1 \in X. \forall \pi_2 \in X. \bigwedge_{a \in \{m_1, \dots, m_n\}} (a_{\pi_1} \leftrightarrow a_{\pi_2}),$$

where $\xi_1 := true \rightarrow \pi \in X$, and $\xi_2$ is defined as

$$\forall \pi_1 \in X. \forall \pi_2 \in sys. \Big( \bigvee_{i \in \{1,\dots,n\}} \boxminus \Big( \bigwedge_{\substack{a \in \{d_1,\dots,d_n\} \cup \\ \{m_1,\dots,m_{i-1}\} \cup \\ \{m_{i+1},\dots,m_n\}}} a_{\pi_1} \leftrightarrow a_{\pi_2} \Big) \Big) \rightarrow \pi_2 \in X.$$

Intuitively, the fixpoint captures all traces that some child $i$ cannot distinguish from $\pi$ in the first $b$ steps, i.e., we add all traces that agree on all APs except $m_i$. Formula $\varphi_{mud}$ then requires that all traces in $X$ agree on the muddiness of all children. The muddy children's MAS violates $\varphi_{mud}$ if and only if $b < n$.

*Scalability.* For the game with $n$ children, we sample random traces and use MoSo to monitor CK (we set the communication-bound to be $b = \lceil \frac{n}{2} \rceil$ so CK does not hold). We report the runtime in Table 2. We observe that even if the number of children grows, our monitor will find violations to common knowledge quickly and

**Table 3: We depict the (average) number of traces the monitor processes before concluding a violation of CK.**

| # children | communication-bound 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| **2** | 1.3 | 7.0 | - | - | - | - |
| **3** | 2.1 | 4.5 | 15.0 | - | - | - |
| **4** | 2.1 | 4.2 | 5.3 | 31.0 | - | - |
| **5** | 2.9 | 3.9 | 5.7 | 33.1 | 63.0 | - |
| **6** | 4.3 | 4.3 | 8.1 | 12.7 | 43.3 | 127.0 |

thus runs very effectively. In contrast, *model-checking* of CK [6] is, currently, only possible up to $n = 4$. This supports our claim that monitoring second-order hyperproperties is a useful lightweight technique that scales in settings where full verification is infeasible.

*Fixpoints vs Second-Order Sets.* The muddy children's puzzle also highlights the importance of fixpoint formulas compared to arbitrary second-order quantification. To test this empirically, we consider the same CK property but express it directly using full second-order quantification (similar to the formula in Section 1). We give the runtime of MoSo on the non-fixpoint-based formula in Table 2 ($t_{noFix}$). When using general second-order quantification, MoSo already times out for 4 children (due to the exponential cost of considering all subsets of traces). This attests to the importance of fixpoints for scalable monitoring.

*Impact of Communication-bound.* Finding violations to the CK formula heavily depends on the communication-bound. With increasing communication-bound, the probability of sampling traces that violate CK decreases as more information is observed by each child. On average, we thus need to see more traces until our monitor reports a violation. We test this empirically by monitoring the muddy children puzzle with a varying number of children and communication-bound and report on the average number of traces (across 10 monitor runs) the monitor observes before concluding that CK does not hold. We depict the results in Table 3. A larger communication bound clearly increases the number of traces the monitor processes until a violation is reported.

## 6.3 Impact of Optimizations

Common knowledge also serves as a useful baseline to highlight the importance of our hashing-based optimizations. To demonstrate this, we check the following CK formula

$$\exists \pi \in sys.\, \Diamond\, fix\Big(X, true \rightarrow \pi \in X, \forall \pi_1 \in X.\, \forall \pi_2 \in sys.\, \big(\boxminus(a_{\pi_1} \leftrightarrow a_{\pi_2}) \vee \boxminus(b_{\pi_1} \leftrightarrow b_{\pi_2})\big) \rightarrow \pi_2 \in X\Big).\, \forall \pi' \in X.\, c_{\pi'},$$

which states that on *some* path $\pi$, $c$ is eventually CK, given that one agent observes AP $a$ and one agent observes AP $b$. We use MoSo to check the above formula on randomly generated instances with a varying number of traces. We depict the results in Figure 3. We observe that all optimizations improve upon the baseline.

## 6.4 Planning Analysis

In our last experiment, we use our monitor to detect reachability in a graph based on the observation of random paths. For example, when observing traces $t_1$ = abc, and $t_2$ = bbd, we can conclude that the trace $t$ = abd exists in the graphs, even without observing it. The inference of $t$ can easily be stated as a fixpoint constraint: We differentiate between time-sensitive (**t-sen.**) and time-insensitive (**t-ins.**) paths. In the former model, we can only "combine" two paths if they visit the same state *at the same time (step)*. In the latter model, we can even combine two paths if they visit the same state at *potentially different times*. We report on the time and number of observed traces it takes to find a correct plan in Table 4. As expected, the time-insensitive case terminates earlier than the time-sensitive case since more combinations are possible, increasing the chance of concluding the existence of a path from source to target.

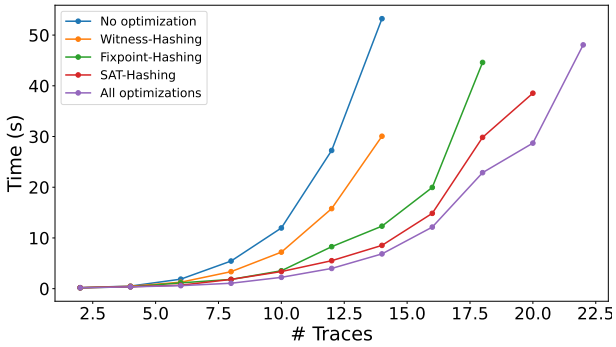


**Figure 3: We compare the optimizations implemented in MoSo as a cactus plot. The timeout is set to 60 sec.**

**Table 4: We monitor the existence of a path from source to target based on random paths in random graphs. For each graph size, we sample 1000 random traces and report the average number of traces that MoSo processes to reach a verdict (# traces) and the average runtime in seconds ($t$).**

| | size | 55 | 70 | 85 | 100 | 115 | 130 |
|---|---|---|---|---|---|---|---|
| **t-sen.** | $t$ | 1.66 | 4.08 | 5.44 | 38.1 | 35.52 | 88.19 |
| | # traces | 1.03 | 1.76 | 3.2 | 3.91 | 4.86 | 5.04 |
| **t-ins.** | $t$ | 45.2 | 61.45 | 72.15 | 112.15 | 76.5 | 137.95 |
| | # traces | 52.15 | 77.35 | 98.25 | 83.95 | 125.75 | 104.55 |

## 7 CONCLUSION

We have presented Hyper$^2$LTL$_f$, the first logic to express second-order hyperproperties for finite trace models. Our experiments show that our monitoring algorithm scales to larger instances well beyond the reach of complete methods such as model checking. Monitoring of complex and important second-order hyperproperties (particularly in the MAS domain), is thus a viable middle ground to obtain rigorous guarantees while maintaining scalability.

## ACKNOWLEDGMENTS

This work was supported by the European Research Council (ERC) Grant HYPER (101055412), by the German Research Foundation (DFG) as part of TRR 248 (389792660), and by the German Israeli Foundation (GIF) Grant No. I-1513-407.2019.

## REFERENCES


[1] Shreya Agrawal and Borzoo Bonakdarpour. 2016. Runtime Verification of k-Safety Hyperproperties in HyperLTL. In *Computer Security Foundations Symposium, CSF 2016*. https://doi.org/10.1109/CSF.2016.24

[2] Suguman Bansal, Yong Li, Lucas M. Tabajara, Moshe Y. Vardi, and Andrew M. Wells. 2023. Model Checking Strategies from Synthesis over Finite Traces. In *International Symposium on Automated Technology for Verification and Analysis, ATVA 2023*. https://doi.org/10.1007/978-3-031-45329-8_11

[3] Andreas Bauer, Martin Leucker, and Christian Schallhart. 2011. Runtime Verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.* (2011). https://doi.org/10.1145/2000799.2000800

[4] Raven Beutner and Bernd Finkbeiner. 2021. A Temporal Logic for Strategic Hyperproperties. In *International Conference on Concurrency Theory, CONCUR 2021*. https://doi.org/10.4230/LIPIcs.CONCUR.2021.24

[5] Raven Beutner and Bernd Finkbeiner. 2024. On Alternating-Time Temporal Logic, Hyperproperties, and Strategy Sharing. In *Conference on Artificial Intelligence, AAAI 2024*.

[6] Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Niklas Metzger. 2023. Second-Order Hyperproperties. In *International Conference on Computer Aided Verification, CAV 2023*. https://doi.org/10.1007/978-3-031-37703-7_15

[7] Borzoo Bonakdarpour and Bernd Finkbeiner. 2018. The Complexity of Monitoring Hyperproperties. In *Computer Security Foundations Symposium, CSF 2018*. https://doi.org/10.1109/CSF.2018.00019

[8] Noel Brett, Umair Siddique, and Borzoo Bonakdarpour. 2017. Rewriting-Based Runtime Verification for Alternation-Free HyperLTL. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2017*. https://doi.org/10.1007/978-3-662-54580-5_5

[9] Alberto Camacho, Jorge A. Baier, Christian J. Muise, and Sheila A. McIlraith. 2018. Finite LTL Synthesis as Planning. In *International Conference on Automated Planning and Scheduling, ICAPS 2018*.

[10] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. 2014. Temporal Logics for Hyperproperties. In *International Conference on Principles of Security and Trust, POST 2014*. https://doi.org/10.1007/978-3-642-54792-8_15

[11] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. 1995. *Reasoning About Knowledge*. MIT Press. https://doi.org/10.7551/mitpress/5803.001.0001

[12] Paolo Felli, Marco Montali, Fabio Patrizi, and Sarah Winkler. 2023. Monitoring Arithmetic Temporal Properties on Finite Traces. In *Conference on Artificial Intelligence, AAAI 2023*. https://doi.org/10.1609/aaai.v37i5.25781

[13] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. 2018. RVHyper: A Runtime Verification Tool for Temporal Hyperproperties. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2018*. https://doi.org/10.1007/978-3-319-89963-3_11

[14] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. 2019. Monitoring hyperproperties. *Formal Methods Syst. Des.* (2019). https://doi.org/10.1007/s10703-019-00334-z

[15] Bernd Finkbeiner and Henny Sipma. 2001. Checking Finite Traces using Alternating Automata. In *Workshop on Runtime Verification, RV 2001*. https://doi.org/10.1016/S1571-0661(04)00250-6

[16] Valeria Fionda and Gianluigi Greco. 2016. The Complexity of LTL on Finite Traces: Hard and Easy Fragments. In *Conference on Artificial Intelligence, AAAI 2016*. https://doi.org/10.1609/aaai.v30i1.10104

[17] Peter Gammie and Ron van der Meyden. 2004. MCK: Model Checking the Logic of Knowledge. In *International Conference on Computer Aided Verification, CAV 2004*. https://doi.org/10.1007/978-3-540-27813-9_41

[18] Giuseppe De Giacomo, Paolo Felli, Marco Montali, and Giuseppe Perelli. 2021. HyperLDLf: a Logic for Checking Properties of Finite Traces Process Logs. In *International Joint Conference on Artificial Intelligence, IJCAI 2021*. https://doi.org/10.24963/IJCAI.2021/256

[19] Giuseppe De Giacomo and Moshe Y. Vardi. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *International Joint Conference on Artificial Intelligence, IJCAI 2013*.

[20] Giuseppe De Giacomo and Moshe Y. Vardi. 2015. Synthesis for LTL and LDL on Finite Traces. In *International Joint Conference on Artificial Intelligence, IJCAI 2015*.

[21] Julian Gutierrez, Giuseppe Perelli, and Michael J. Wooldridge. 2021. Multi-player games with LDL goals over finite traces. *Inf. Comput.* (2021). https://doi.org/10.1016/J.IC.2020.104555

[22] Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. 2020. Propositional Dynamic Logic for Hyperproperties. In *International Conference on Concurrency Theory, CONCUR 2020*. https://doi.org/10.4230/LIPIcs.CONCUR.2020.50

[23] Joseph Y. Halpern and Yoram Moses. 1990. Knowledge and Common Knowledge in a Distributed Environment. *J. ACM* 37, 3 (1990), 549–587. https://doi.org/10.1145/79147.79161

[24] Klaus Havelund and Grigore Rosu. 2004. Efficient monitoring of safety properties. *Int. J. Softw. Tools Technol. Transf.* (2004). https://doi.org/10.1007/s10009-003-0117-6

[25] Jeremy Kong and Alessio Lomuscio. 2017. Model Checking Multi-Agent Systems against LDLK Specifications. In *International Joint Conference on Artificial Intelligence, IJCAI 2017*. https://doi.org/10.24963/ijcai.2017/158

[26] Jeremy Kong and Alessio Lomuscio. 2018. Model Checking Multi-Agent Systems against LDLK Specifications on Finite Traces. In *International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018*.

[27] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. 2017. MCMAS: an open-source model checker for the verification of multi-agent systems. *Int. J. Softw. Tools Technol. Transf.* (2017). https://doi.org/10.1007/s10009-015-0378-x

[28] Nicolas Markey. 2003. Temporal logic with past is exponentially more succinct, Concurrency Column. *Bull. EATCS* (2003).

[29] Markus N. Rabe. 2016. *A temporal logic approach to information-flow control*. Ph.D. Dissertation. Saarland University.

[30] Senthil Rajasekaran and Moshe Y. Vardi. 2022. Verification and Realizability in Finite-Horizon Multiagent Systems. In *International Conference on Principles of Knowledge Representation and Reasoning, KR 2022*.

[31] Alfred Tarski. 1955. A lattice-theoretical fixpoint theorem and its applications. (1955).

[32] Ron van der Meyden. 1998. Common Knowledge and Update in Finite Environments. *Inf. Comput.* (1998). https://doi.org/10.1006/inco.1997.2679