Monitoring Hyperproperties[⋆]

Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup

Reactive Systems Group
Saarland University
`lastname@react.uni-saarland.de`

Abstract. We investigate the runtime verification problem of hyperproperties, such as non-interference and observational determinism, given as formulas of the temporal logic HyperLTL. HyperLTL extends linear-time temporal logic (LTL) with trace quantifiers and trace variables. We show that deciding whether a HyperLTL formula is monitorable is PSPACE-complete. For monitorable specifications, we present an efficient monitoring approach. As hyperproperties relate multiple computation traces with each other, it is necessary to store previously seen traces, and to relate new traces to the traces seen so far. If done naively, this causes the monitor to become slower and slower, before it inevitably runs out of memory. In this paper, we present techniques that reduce the set of traces that new traces must be compared against to a minimal subset. Additionally, we exploit properties of specifications such as reflexivity, symmetry, and transitivity, to reduce the number of comparisons. We show that this leads to much more scalable monitoring with, in particular, significantly lower memory consumption.

1 Introduction

Hyperproperties [10] generalize trace properties in that they not only check the correctness of individual traces, but can also relate multiple computation traces to each other. This is needed, for example, to express information flow security policies like the requirement that the system behavior appears to be deterministic, i.e., independent of certain secrets, to an external observer. Monitoring hyperproperties is difficult, because it is no longer possible to analyze traces in isolation: a violation of a hyperproperty in general involves a set of traces, not just a single trace. A naive approach would be to simply store all traces seen so far. This would create two problems: a memory problem, because the needed memory grows with the number of traces observed by the monitor, and a time problem, because one needs to relate every newly observed trace against the growing set of stored traces.

There are hyperproperties where this effect cannot be avoided. An example is the hyperproperty with two atomic propositions p and q, where any pair of

⋆ This work was partially supported by the German Research Foundation (DFG) under the project SpAGAT (grant no. FI 936/2-1) in the priority program "Reliably Secure Software Systems – RS3" and as part of the Collaborative Research Center "Methods and Tools for Understanding and Controlling Privacy" (SFB 1223).

traces that agree on their p labeling must also agree on their q labeling. Clearly, for every p labeling seen so far, we must also store the corresponding q labeling. In practice, however, it is often possible to greatly simplify the monitoring. Consider, for example, the hyperproperty that states that all traces have the same q labeling (independently of the p labeling). In the temporal logic HyperLTL [9], this property is specified as the formula $\forall \pi. \forall \pi'. \square(p_\pi \leftrightarrow p_{\pi'})$. The naive approach would store all traces seen so far, and thus require $O(n)$ memory after n traces. A new trace would be compared against every stored trace twice, once as π and once as π', resulting in a $O(2n)$ running time for each new trace. Obviously, however, it is sufficient in this example to store the first trace, and compare all further incoming traces against this reference. The required memory is thus, in fact, constant in the number of traces. A further observation is that the specification is symmetric in π and π'. Hence, a single comparison suffices.

In this paper, we present a monitoring approach for hyperproperties that reduces the set of traces that new traces must be compared against to a minimal subset. Our approach comes with a strong correctness guarantee: our monitor produces the same verdict as a naive monitor that would store all traces and, additionally, we keep a sufficient set of traces to always provide an actually observed witness for the monitoring verdict. Our monitoring thus delivers a result that is equally informative as the naive solution, but is computed faster and with less memory.

We introduce two analysis techniques: *Trace analysis* reduces the stored set of traces to a minimum, thus minimizing the required memory. *Specification analysis* identifies symmetry, transitivity, and reflexivity in the specification, in order to reduce the algorithmic workload that needs to be carried out on the stored traces.

Trace Analysis. As an example for a system where confidentiality and information flow is of outstanding importance for the intended operation, we consider a conference management system. There are a number of confidentiality properties that such a system should satisfy, like *"The final decision of the program committee remains secret until the notification"* and *"All intermediate decisions of the program committee are never revealed to the author"*. We want to focus on important hyperproperties of interest beyond confidentiality, like the property that no paper submission is lost or delayed. Informally, one formulation of this property is *"A paper submission is immediately visible for every program committee member"*. More formally, this property relates pairs of traces, one belonging to an author and one belonging to a program committee member. We assume this separation is indicated by a proposition pc that is either disabled or enabled in the first component of those traces. Further propositions in our example are the proposition s, denoting that a paper has been submitted, and v denoting that the paper is visible.

Given a set of traces T, we can verify that the property holds by checking every pair of traces $(t, t') \in T \times T$ with $pc \notin t[0]$ and $pc \in t'[0]$ that $s \in t[i]$ implies $v \in t'[i+1]$ for every $i \geq 0$. When T satisfies the property, $T \cup \{t^*\}$, where t^* is a new trace, amounts to checking new pairs (t^*, t) and (t, t^*) for $t \in T$. This, however, leads to an increasing size of T and thereby to an increased number of checks: the monitoring problem becomes inevitable costlier over time. To circumvent this, we present a method that keeps the set of traces *minimal* with respect to the underlying property. When monitoring hyperproperties, traces may pose *requirements* on future traces. The core idea of our approach is to characterize traces that pose strictly stronger requirements on future traces than others. In this case, the traces with the weaker requirements can be safely discarded. As an example, consider the following set of traces

{s}	{}	{}	{}	{}	*an author immediately submits a paper*	(1)
{}	{s}	{}	{}	{}	*an author submits a paper after one time unit*	(2)
{}	{s}	{s}	{}	{}	*an author submits two papers*	(3)

A satisfying PC trace would be $\{pc\}\{v\}\{v\}\{v\}\emptyset$ as there are author traces with paper submissions at time step 0, 1, and 2. For checking our property, one can safely discard trace 2 as it poses no more requirements than trace 3. We say that trace 3 dominates trace 2. We show that, given a property in the temporal logic HyperLTL, we can automatically reduce trace sets to be minimal with respect to this dominance. On relevant and more complex information flow properties, this reduces the memory consumption dramatically.

Specification Analysis. For expressing hyperproperties, we use the recently introduced temporal logic HyperLTL [9], which extends linear-time temporal logic (LTL) [22] with explicit trace quantification. We construct a monitor template, containing trace variables, from the HyperLTL formula. We initialize this monitor with explicit traces resulting in a family of monitors checking the relation, defined by the hyperoperty, between the traces. Our specification analysis technique allows us to reduce the number of monitors in order to detect violation or satisfaction of a given HyperLTL formula. We use the decision procedure for the satisfiability problem of HyperLTL [15] to check whether or not a universally quantified HyperLTL formula is symmetric, transitive, or reflexive. If a hyperproperty is *symmetric*, then we can omit every symmetric monitor, thus, performing only half of the language membership tests. A canonical example for a symmetric HyperLTL formula is $\mathit{ObsDet} \coloneqq \forall \pi. \forall \pi'.\ (O_\pi = O_{\pi'}) \mathcal{W} (I_\pi \neq I_{\pi'})$, a variant of observational determinism [21,24,30]. Symmetry is particular interesting, since many information flow policies have this property. If a hyperproperty is *transitive*, then we can omit every, except for one, monitor, since we can check every incoming trace against any reference trace. One example for a transitive HyperLTL formula is equality $EQ \coloneqq \forall \pi. \forall \pi'. \square(a_\pi \leftrightarrow a_{\pi'})$. If a hyperproperty is *reflexive*, then we can omit the monitor where every trace variable is initialized with the same trace. For example, both hyperproperties above are reflexive.

Related Work. The temporal logic HyperLTL was introduced to model check security properties of reactive systems [9, 17]. For one of its predecessors, SecLTL [13], there has been a proposal for a white box monitoring approach [14] based on alternating automata. The problem of monitoring HyperLTL has been considered before [1,7]. Agrawal and Bonakdarpour [1] gave a syntactic characterization of monitorable HyperLTL formulas and a monitoring algorithm based on Petri nets. In subsequent work, a constraint based approach has been proposed [7]. Like our monitoring algorithm, they do not have access to the implementation (black box), but in contrast to our work, they do not provide witnessing traces for a monitor verdict. For certain information flow policies, like non-interference and some extensions, dynamic enforcement mechanisms have been proposed. Techniques for the enforcement of information flow policies include tracking dependencies at the hardware level [27], language-based monitors [2,3,5,25,29], and abstraction-based dependency tracking [8,18,19]. Secure multi-execution [12] is a technique that can enforce non-interference by executing a program multiple times in different security levels. To enforce non-interference, the inputs are replaced by default values whenever a program tries to read from a higher security level.

2 Monitorability of HyperLTL

Let AP be a finite set of atomic propositions and let $\Sigma = 2^{\mathrm{AP}}$ be the corresponding finite *alphabet*. A finite (infinite) trace is a finite (infinite) sequence over Σ. We denote the concatenation of a finite trace $u \in \Sigma^*$ and a finite or infinite trace $v \in \Sigma^* \cup \Sigma^\omega$ by uv and write $u \preceq v$ if u is a prefix of v. Further, we lift the prefix operator to sets of traces, i.e., $U \preceq V := \forall u \in U. \exists v \in V. u \preceq v$ for $U \subseteq \Sigma^*$ and $V \subseteq \Sigma^* \cup \Sigma^\omega$. We denote the powerset of a set A by $\mathcal{P}(A)$ and define $\mathcal{P}^*(A)$ to be the set of all finite subsets of A.

HyperLTL. HyperLTL [9] is a temporal logic for specifying hyperproperties. It extends LTL [22] by quantification over trace variables π and a method to link atomic propositions to specific traces. The set of trace variables is $\mathcal{V}$. Formulas in HyperLTL are given by the grammar

$$\varphi ::= \forall \pi. \varphi \mid \exists \pi. \varphi \mid \psi \ , \text{ and}$$
$$\psi ::= a_\pi \mid \neg\psi \mid \psi \vee \psi \mid \bigcirc \psi \mid \psi \,\mathcal{U}\, \psi \ ,$$

where $a \in \mathrm{AP}$ and $\pi \in \mathcal{V}$. We call a HyperLTL formula an LTL formula if it is quantifier free. The semantics is given by the satisfaction relation $\vDash_P$ over a set of traces $T \subseteq \Sigma^\omega$. We define an assignment $\Pi : \mathcal{V} \to \Sigma^\omega$ that maps trace variables to traces. $\Pi[i, \infty]$ denotes the trace assignment that is equal to $\Pi(\pi)[i, \infty]$ for all π.

$\Pi \vDash_T a_\pi$	if $a \in \Pi(\pi)[0]$
$\Pi \vDash_T \neg\varphi$	if $\Pi \nvDash_T \varphi$
$\Pi \vDash_T \varphi \vee \psi$	if $\Pi \vDash_T \varphi$ or $\Pi \vDash_T \psi$
$\Pi \vDash_T \bigcirc \varphi$	if $\Pi[1, \infty] \vDash_T \varphi$
$\Pi \vDash_T \varphi \,\mathcal{U}\, \psi$	if $\exists i \geq 0. \Pi[i, \infty] \vDash_T \psi \wedge \forall 0 \leq j < i. \Pi[j, \infty] \vDash_T \varphi$
$\Pi \vDash_T \exists \pi. \varphi$	if there is some $t \in T$ such that $\Pi[\pi \mapsto t] \vDash_T \varphi$

We write $T \vDash \varphi$ for $\{\} \vDash_T \varphi$ where $\{\}$ denotes the empty assignment. The language of a HyperLTL formula φ, denoted by $\mathcal{L}(\varphi)$, is the set $\{T \subseteq \Sigma^\omega \mid T \vDash \varphi\}$. Let φ be a HyperLTL formula with trace variables $\mathcal{V} = \{\pi_1, \dots, \pi_k\}$ over alphabet Σ. We define $\Sigma_\mathcal{V}$ to be the alphabet where p_π is interpreted as an atomic proposition for every $p \in \mathrm{AP}$ and $\pi \in \mathcal{V}$. We denote by $\vDash_{\mathrm{LTL}}$ the LTL satisfaction relation over $\Sigma_\mathcal{V}$. We define the π-projection, denoted by $\#_\pi(s)$, for a given $s \subseteq \Sigma_\mathcal{V}$ and $\pi \in \mathcal{V}$, as the set of all $p_\pi \in s$.

Lemma 1. *Let ψ be an LTL formula over trace variables $\mathcal{V}$. There is a trace assignment A such that $A \vDash_\emptyset \psi$ if, and only if, ψ is satisfiable under LTL semantics over atomic propositions $\Sigma_\mathcal{V}$. The models can be translated effectively.*

Monitorability. For the remainder of this section, we develop the notion of monitorability for hyperproperties and show that deciding whether a HyperLTL formula is monitorable is PSpace-complete, i.e., no harder than the corresponding problem for LTL. This result extends earlier characterizations based on restricted syntactic fragments of HyperLTL [1].

For trace languages, monitorability is the property whether language containment can be decided by finite prefixes [23]. Given a trace language $L \subseteq \Sigma^\omega$, the set of *good* and *bad* prefixes is $\mathit{good}(L) \coloneqq \{u \in \Sigma^* \mid \forall v \in \Sigma^\omega.\, uv \in L\}$ and $\mathit{bad}(L) \coloneqq \{u \in \Sigma^* \mid \forall v \in \Sigma^\omega.\, uv \notin L\}$, respectively. L is *monitorable* if $\forall u \in \Sigma^*.\, \exists v \in \Sigma^*.\, uv \in \mathit{good}(L) \vee uv \in \mathit{bad}(L)$. The decision problem, i.e., given an LTL formula φ, decide whether φ is monitorable, is PSpace-complete [4].

A *hyperproperty* H is a set of trace properties, i.e., $H \subseteq \mathcal{P}(\Sigma^\omega)$. Analogous to the previous definition, we define monitorability for hyperproperties. Given $H \subseteq \mathcal{P}(\Sigma^\omega)$. The set of *good* and *bad prefix traces* is $\mathit{good}(H) \coloneqq \{U \in \mathcal{P}^*(\Sigma^*) \mid \forall V \in \mathcal{P}(\Sigma^\omega).\, U \preceq V \Rightarrow V \in H\}$ and $\mathit{bad}(H) \coloneqq \{U \in \mathcal{P}^*(\Sigma^*) \mid \forall V \in \mathcal{P}(\Sigma^\omega).\, U \preceq V \Rightarrow V \notin H\}$, respectively. H is *monitorable* if

$$\forall U \in \mathcal{P}^*(\Sigma^*).\, \exists V \in \mathcal{P}^*(\Sigma^*).\, U \preceq V \Rightarrow V \in \mathit{good}(H) \vee V \in \mathit{bad}(H) \ .$$

We present a method to decide whether an alternation-free HyperLTL formula is monitorable.

Lemma 2. *Given a HyperLTL formula $\varphi = \forall \pi_1 \dots \forall \pi_k.\, \psi$, where ψ is an LTL formula. It holds that $\mathit{good}(\mathcal{L}(\varphi)) = \emptyset$ unless $\psi \equiv \mathit{true}$.*

Theorem 1. *Given a HyperLTL formula $\varphi = \forall \pi_1 \dots \forall \pi_k.\, \psi$, where $\psi \not\equiv \mathit{true}$ is an LTL formula. φ is monitorable if, and only if, $\forall u \in \Sigma_\mathcal{V}^*.\, \exists v \in \Sigma_\mathcal{V}^*.\, uv \in \mathit{bad}(\mathcal{L}(\psi))$.*

Proof. Assume $\forall u \in \Sigma_\mathcal{V}^*.\, \exists v \in \Sigma_\mathcal{V}^*.\, uv \in \mathit{bad}(\mathcal{L}(\psi))$ holds. Given an arbitrary prefix $U \in \mathcal{P}^*(\Sigma^*)$. Pick an arbitrary mapping from U to $\Sigma_\mathcal{V}^*$ and call it u'. By assumption, there is a $v' \in \Sigma_\mathcal{V}^*$ such that $u'v' \in \mathit{bad}(\mathcal{L}(\psi))$. We use this v' to extend the corresponding traces in U resulting in $V \in \mathcal{P}^*(\Sigma^*)$. It follows that for all $W \in \mathcal{P}(\Sigma^\omega)$ with $V \preceq W$, $W \nvDash \varphi$, hence, $V \in \mathit{bad}(\mathcal{L}(\varphi))$.

Assume φ is monitorable, thus, $\forall U \in \mathcal{P}^*(\Sigma^*).\, \exists V \in \mathcal{P}^*(\Sigma^*).\, U \preceq V \Rightarrow V \in \mathit{good}(\mathcal{L}(\varphi)) \vee V \in \mathit{bad}(\mathcal{L}(\varphi))$. As the set of good prefixes $\mathit{good}(\mathcal{L}(\varphi))$ is empty by Lemma 2 we can simplify the formula to $\forall U \in \mathcal{P}^*(\Sigma^*).\, \exists V \in \mathcal{P}^*(\Sigma^*).\, U \preceq V \Rightarrow V \in \mathit{bad}(\mathcal{L}(\varphi))$. Given an arbitrary $u \in \Sigma_{\mathcal{V}}^*$, we translate it into the (canonical) U' and get a V' satisfying the conditions above. Let $v' \in \Sigma_{\mathcal{V}}^*$ be the finite trace constructed from the extensions of u in V' (not canonical, but all are bad prefixes since $V' \in \mathit{bad}(\mathcal{L}(\varphi))$). By assumption, $u'v' \in \mathit{bad}(\mathcal{L}(\psi))$. $\square$

Corollary 1. *Given a HyperLTL formula $\varphi = \exists\pi_1 \ldots \exists\pi_k.\psi$, where ψ is an LTL formula. φ is monitorable if, and only if, $\forall u \in \Sigma_{\mathcal{V}}^*.\, \exists v \in \Sigma_{\mathcal{V}}^*.\, uv \in \mathit{good}(\mathcal{L}(\psi))$.*

Theorem 2. *Given an alternation-free HyperLTL formula φ. Deciding whether φ is monitorable is* PSPACE*-complete.*

Proof. We consider the case that $\varphi = \forall\pi_1 \ldots \forall\pi_2.\, \psi$, the case for existentially quantified formulas is dual. We apply the characterization from Theorem 1. First, we have to check validity of ψ which can be done in polynomial space [26]. Next, we have to determine whether $\forall u \in \Sigma_{\mathcal{V}}^*.\, \exists v \in \Sigma_{\mathcal{V}}^*.\, uv \in \mathit{bad}(\mathcal{L}(\psi))$. We use a slight modification of the PSPACE algorithm given by Bauer [4]. Hardness follows as the problem is already PSPACE-hard for LTL. $\square$

3 Monitoring HyperLTL

There are many obstacles to overcome in monitoring hyperproperties (see [6] for an overview of the challenges), such that classic monitoring approaches of trace properties need to be carefully adjusted. In this section, we define a finite trace semantics for HyperLTL and present our automata-based monitoring approach.

Finite Trace Semantics. We define a finite trace semantics for HyperLTL based on the finite trace semantics of LTL [20]. In the following, when using $\mathcal{L}(\varphi)$ we refer to the finite trace semantics of a HyperLTL formula φ. Let t be a finite trace, ϵ denotes the empty trace, and $|t|$ denotes the length of a trace. Since we are in a finite trace setting, $t[i, \ldots]$ denotes the subsequence from position i to position $|t|-1$. Let $\Pi_{\mathit{fin}} : \mathcal{V} \to \Sigma^*$ be a partial function mapping trace variables to finite traces. We define $\epsilon[0]$ as the empty set. $\Pi_{\mathit{fin}}[i, \ldots]$ denotes the trace assignment that is equal to $\Pi_{\mathit{fin}}(\pi)[i, \ldots]$ for all π. We define a subsequence of t as follows.

$$t[i,j] = \begin{cases} \epsilon & \text{if } i \geq |t| \\ t[i, \mathit{min}(j, |t|-1)], & \text{otherwise} \end{cases}$$

$$\begin{array}{ll}
\Pi_{\mathit{fin}} \vDash_T a_\pi & \text{if } a \in \Pi_{\mathit{fin}}(\pi)[0] \\
\Pi_{\mathit{fin}} \vDash_T \neg\varphi & \text{if } \Pi_{\mathit{fin}} \nvDash_T \varphi \\
\Pi_{\mathit{fin}} \vDash_T \varphi \vee \psi & \text{if } \Pi_{\mathit{fin}} \vDash_T \varphi \text{ or } \Pi_{\mathit{fin}} \vDash_T \psi \\
\Pi_{\mathit{fin}} \vDash_T \bigcirc\varphi & \text{if } \Pi_{\mathit{fin}}[1, \ldots] \vDash_T \varphi \\
\Pi_{\mathit{fin}} \vDash_T \varphi\,\mathcal{U}\,\psi & \text{if } \exists i \geq 0.\, \Pi_{\mathit{fin}}[i, \ldots] \vDash_T \psi \wedge \forall 0 \leq j < i.\, \Pi_{\mathit{fin}}[j, \ldots] \vDash_T \varphi \\
\Pi_{\mathit{fin}} \vDash_T \exists\pi.\, \varphi & \text{if there is some } t \in T \text{ such that } \Pi_{\mathit{fin}}[\pi \mapsto t] \vDash_T \varphi
\end{array}$$

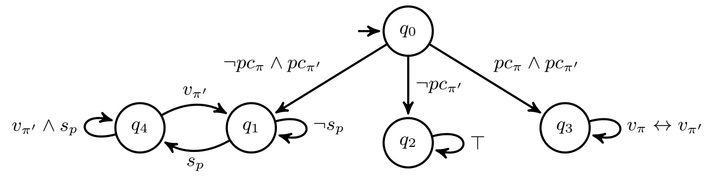

Fig. 1. Visualization of a monitor template corresponding to formula given in Equation 4. We use a symbolic representation of the transition function δ.

Monitoring Algorithm. In this subsection, we describe our automata-based monitoring algorithm for HyperLTL. We employ standard techniques for building LTL monitoring automata and use this to instantiate this monitor by the traces as specified by the HyperLTL formula.

Let AP be a set of atomic propositions and $\mathcal{V} = \{\pi_1, \ldots, \pi_n\}$ a set of trace variables. A deterministic monitor template $\mathcal{M} = (\Sigma, Q, \delta, q_0)$ is a four tuple of a finite alphabet $\Sigma = 2^{\mathrm{AP} \times \mathcal{V}}$, a non-empty set of states Q, a partial transition function $\delta : Q \times \Sigma \to Q$, and a designated initial state $q_0 \in Q$. The automata runs in parallel over traces $(2^{\mathrm{AP}})^*$, thus we define a run with respect to a n-ary tuple $N \in ((2^{\mathrm{AP}})^*)^n$ of finite traces. A run of N is a sequence of states $q_0 q_1 \cdots q_m \in Q^*$, where m is the length of the smallest trace in N, starting in the initial state q_0 such that for all i with $0 \leq i < m$ it holds that

$$\delta\left(q_i, \bigcup_{j=1}^{n} \bigcup_{a \in N(j)(i)} \{(a, \pi_j)\}\right) = q_{i+1} \quad .$$

A tuple N is accepted, if there is a run on $\mathcal{M}$. For LTL, such a deterministic monitor can be constructed in doubly-exponential time in the size of the formula [11, 28].

Example 1. We consider again the conference management example from the introduction. We distinguish two types of traces, *author traces* and *program committee member traces*, where the latter starts with proposition *pc*. Based on this traces, we want to verify that no paper submission is lost, i.e., that every submission (proposition s) is visible (proposition v) to every program committee member in the following step. When comparing two PC traces, we require that they agree on proposition v. The monitor template for the following HyperLTL formalization is depicted in Fig. 1.

$$\forall \pi. \forall \pi'. \big((\neg pc_\pi \wedge pc_{\pi'}) \to \bigcirc\Box(s_\pi \to \bigcirc v_{\pi'})\big) \wedge \big((pc_\pi \wedge pc_{\pi'}) \to \bigcirc\Box(v_\pi \leftrightarrow v'_\pi)\big) \quad (4)$$

The offline and online algorithms for monitoring HyperLTL formulas are presented in Fig. 2. The *offline* algorithm takes a HyperLTL formula φ and a set of traces T as input. After building the deterministic monitoring automaton $\mathcal{M}_\varphi$, it checks every n-ary tuple $N \in T^n$. If some trace tuple N is not accepted by $\mathcal{M}_\varphi$, then this path assignment violates the formula φ. The *online* algorithm is similar, but proceeds with the pace of the incoming stream, which has an

```
input  : $\forall^n$ HyperLTL formula $\varphi$
         set of traces $T$
output: satisfied or $n$-ary tuple
         witnessing violation
$\mathcal{M}_\varphi$ = build_template($\varphi$);
for each tuple $N \in T^n$ do
    if $\mathcal{M}_\varphi$ accepts $N$ then
        proceed;
    else
        return $N$;
    end
end
return satisfied;
```

Algorithm 1: Offline Algorithm.

```
input  : $\forall^n$ HyperLTL formula $\varphi$
output: satisfied or $n$-ary tuple
         witnessing violation
$\mathcal{M}_\varphi = (\Sigma, Q, \delta, q_0)$ = build_template($\varphi$);
$S : T^n \to Q$;
$T := \emptyset$;
$t := \epsilon$;
while $p \leftarrow$ new element do
    if $p$ is new trace then
        $T \cup \{t\}$;
        $t := \epsilon$;
        $S := \{q_0 \mid$ for new $n$-tuple$\}$;
    else
        $t := t\ p$;
        progress every state in $S$ according to $\delta$;
        if violation then
            return witnessing tuple;
        end
    end
end
return satisfied;
```

Algorithm 2: Online Algorithm.

Fig. 2. Evaluation algorithms for monitoring $\forall^n$ HyperLTL formulas.

indicator when a new trace starts. We have a variable S that maps tuples of traces to states of the deterministic monitor. Whenever a trace progresses, we update the states in S according to the transition function δ. If on this progress, there is a violation, we return the corresponding tuple of traces as a witness. When a new trace t starts, only new tuples are considered for S, that are tuples $N \in (T \cup \{t\})^n$ containing the new trace t, i.e., $N \notin T^n$.

In contrast to previous approaches, our algorithm returns a witness for violation. This highly desired property comes with a price. In constructed worst case scenarios, we have to remember every system trace in order to return an explicit witness. However, it turns out that practical hyperproperties satisfy certain properties such that the majority of traces can be pruned during the monitoring process.

4 Minimizing Trace Storage

The main obstacle in monitoring hyperproperties is the potentially unbounded space consumption. In the following, we present two analysis phases of our algorithm. The first phase is a specification analysis, which is a preprocessing step that analyzes the HyperLTL formula under consideration. We use the recently introduced satisfiability solver for hyperproperties EAHyper [16] to detect whether a formula is (1) *symmetric*, i.e., we halve the number of instantiated monitors, (2) *transitive*, i.e, we reduce the number of instantiated monitors to two, or (3) *reflexive*, i.e., we can omit the self comparison of traces. The second analysis phase is applied during runtime. We analyze the incoming trace to detect whether or not this trace poses strictly more requirements on future traces, with respect to a given HyperLTL formula.

4.1 Specification Analysis

Symmetry. Symmetry is particular interesting since many information flow policies satisfy this property. Consider, for example, observational determinism $\mathit{ObsDet} \coloneqq \forall\pi.\,\forall\pi'.\,(O_\pi = O_{\pi'})\,\mathcal{W}\,(I_\pi \neq I_{\pi'})$. We detect symmetry by translating this formula to a formula ObsDet_{symm} that is unsatisfiable if there exists no set of traces for which every trace pair violates the symmetry condition:

$$\mathit{ObsDet}_{symm} \coloneqq \exists\pi.\,\exists\pi'.\,\big((O_\pi = O_{\pi'})\mathcal{W}(I_\pi \neq I_{\pi'})\big) \nleftrightarrow \big((O'_\pi = O_\pi)\mathcal{W}(I'_\pi \neq I_\pi)\big)$$

This is a sufficient condition for the invariance of ObsDet under π and π', which we define in the following, and, therefore, ObsDet is symmetric.

Definition 1. *Given a HyperLTL formula $\varphi = \forall\pi_1 \ldots \forall\pi_n.\,\psi$, where ψ is an LTL formula over trace variables $\{\pi_1, \ldots, \pi_n\}$. We say φ is invariant under trace variable permutation $\sigma : \mathcal{V} \to \mathcal{V}$, if for any set of traces $T \subseteq \Sigma^\omega$ and any assignment $\Pi : \mathcal{V} \to T$, $\Pi \vDash_T \psi \Leftrightarrow (\Pi \circ \sigma) \vDash_T \psi$. We say φ is symmetric, if it is invariant under every trace variable permutation in $\mathcal{V} \to \mathcal{V}$.*

We generalize the previous example to formulas with more than two universal quantifiers. We use the fact, that the symmetric group for a finite set $\mathcal{V}$ of n trace variables is generated by the two permutations $(\pi_1\ \pi_2)$ and $(\pi_1\ \pi_2\ \cdots\ \pi_{n-1}\ \pi_n)$. If the HyperLTL-SAT solver determines that the input formula is invariant under these two permutations, then the formula is invariant under every trace variable permutation and thus symmetric.

Theorem 3. *Given a HyperLTL formula $\varphi = \forall\pi_1 \ldots \forall\pi_n.\,\psi$, where ψ is an LTL formula over trace variables $\{\pi_1, \ldots, \pi_n\}$. φ is symmetric if and only if $\varphi_{symm} = \exists\pi_1 \ldots \exists\pi_n.\,(\psi(\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n) \nleftrightarrow \psi(\pi_2, \pi_1, \ldots, \pi_{n-1}, \pi_n))$ $\vee(\psi(\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n) \nleftrightarrow \psi(\pi_2, \pi_3, \ldots, \pi_n, \pi_1))$ is unsatisfiable.*

Transitivity. While symmetric HyperLTL formulas allow us to prune half of the monitor instances, transitivity of a HyperLTL formula has an even larger impact on the required memory. Observational Determinism, considered above, is not transitive. However, equality, i.e, $EQ \coloneqq \forall\pi.\forall\pi'.\Box(a_\pi \leftrightarrow a_{\pi'})$, for example, is transitive and symmetric and allow us to reduce the number of monitor instances to one, since we can check equality against any reference trace.

Definition 2. *Given a HyperLTL formula $\varphi = \forall\pi_1.\,\forall\pi_2.\,\psi$, where ψ is an LTL formula over trace variables $\{\pi_1, \pi_2\}$. Let $T = \{t_1, t_2, t_3\} \in \Sigma^\omega$ be three-elemented set of traces. We define the assignment $\Pi_{i,j} : \mathcal{V} \to \Sigma^\omega$ by $\Pi_{i,j} \coloneqq \{\pi_1 \mapsto t_i, \pi_2 \mapsto t_j\}$. We say φ is transitive, if T was chosen arbitrary and $(\Pi_{1,2} \vDash_T \psi) \wedge (\Pi_{2,3} \vDash_T \psi) \Rightarrow \Pi_{1,3} \vDash_T \psi$.*

Theorem 4. *Given a HyperLTL formula $\varphi = \forall\pi_1.\,\forall\pi_2.\,\psi$, where ψ is an LTL formula over trace variables $\{\pi_1, \pi_2\}$. φ is transitive if and only if $\varphi_{trans} = \exists\pi_1\exists\pi_2\exists\pi_3.\,(\psi(\pi_1, \pi_2) \wedge \psi(\pi_2, \pi_3)) \nrightarrow \psi(\pi_1, \pi_3)$ is unsatisfiable.*

Reflexivity. Lastly, we introduce a method to check whether a formula is reflexive, which enables us to omit the composition of a trace with itself in the monitoring algorithm. Both HyperLTL formulas considered in this section, *ObsDet* and *EQ*, are reflexive.

Definition 3. *Given a HyperLTL formula* $\varphi = \forall \pi_1 \dots \forall \pi_n .\, \psi$*, where* ψ *is an LTL formula over trace variables* $\{\pi_1, \dots, \pi_n\}$*. We say* φ *is reflexive, if for any trace* $t \in \Sigma^\omega$ *and the corresponding assignment* $\Pi : \mathcal{V} \to \{t\}$*,* $\Pi \vDash_{\{t\}} \psi$*.*

Theorem 5. *Given a HyperLTL formula* $\varphi = \forall \pi_1 \dots \forall \pi_n .\, \psi$*, where* ψ *is an LTL formula over trace variables* $\{\pi_1, \dots, \pi_n\}$*.* φ *is reflexive if and only if* $\varphi_{\mathit{refl}} = \exists \pi .\, \neg \psi(\pi, \pi, \dots, \pi)$ *is unsatisfiable.*

4.2 Trace Analysis

In the previous subsection, we described a preprocessing step to reduce the number of monitor instantiations. The main idea of the trace analysis, considered in the following, is to check whether a trace contains new requirements on the system under consideration. If this is not the case, then this trace will not be stored by our monitoring algorithm. We denote $\mathcal{M}_\varphi$ as the monitor template of a $\forall^*$ HyperLTL formula φ.

Definition 4. *Given a HyperLTL formula* φ*, a trace set* T *and an arbitrary* $t \in TR$*, we say that* t *is* (T, φ)*-redundant if* T *is a model of* φ *if and only if* $T \cup \{t\}$ *is a model of* φ *as well. Formally denoted as follows.*

$$\forall T' \supseteq T .\, T' \in \mathcal{L}(\varphi) \Leftrightarrow T' \cup \{t\} \in \mathcal{L}(\varphi).$$

Example 2. Consider, again, our example hyperproperty for a conference management system. *"A user submission is immediately visible for every program committee member and every program committee member observes the same."* We formalized this property as a $\forall^2$ HyperLTL formula in Equation 4. Assume our algorithm observes the following three traces of length five.

{}	{s}	{}	{}	{}	*an author submits a paper*	(5)
{}	{}	{s}	{}	{}	*an author submits a paper one time unit later*	(6)
{}	{}	{s}	{s}	{}	*an author submits two papers*	(7)

Trace 6 contains, with respect to φ above, no more information than trace 7. We say that trace 7 dominates trace 6 and, hence, trace 6 may be pruned from the set of traces that the algorithm has to store. If we consider a PC member trace, we encounter the following situation.

{}	{s}	{}	{}	{}	*an author submits a paper*	(8)
{}	{}	{s}	{s}	{}	*an author submits two papers*	(9)
{}	{pc}	{v}	{v}	{v}	*a PC member observes three submissions*	(10)

Our algorithm will detect no violation, since the program committee member sees all three papers. Intuitively, one might expect that no more traces can be pruned from this trace set. However, in fact, trace 10 dominates trace 8 and trace 9, since the information that three papers have been submitted is preserved in trace 10. Hence, it suffices to remember the last trace to detect, for example, the following violations.

$$\boxed{\{\}}\boxed{\{\text{pc}\}}\boxed{\{\text{v}\}}\boxed{\{\text{v}\}}\boxed{\{\text{v}\}} \quad \textit{a PC member observes three submissions} \tag{11}$$

$$\boxed{\{\}}\boxed{\{\text{pc}\}}\boxed{\{\text{v}\}}\boxed{\{\text{v}\}}\boxed{\{\}} \quad \lightning\,\textit{a PC member observes two submissions}\,\lightning \tag{12}$$

or

$$\boxed{\{\}}\boxed{\{\}}\boxed{\{\}}\boxed{\{\}}\boxed{\{\text{s}\}} \quad \lightning\,\textit{an author submits a non-visible paper}\,\lightning \tag{13}$$

Note that none of the previous user traces, i.e., trace 5 to trace 9, are needed to detect a violation.

Definition 5. *Given $t, t' \in TR$, we say t* dominates *t' if t' is $(\{t\}, \varphi)$-redundant.*

The observations from Example 2 can be generalized to a language inclusion check (cf. Theorem 6), to determine whether a trace dominates another trace. For proving this, we first prove the following two lemmas. For the sake of simplicity, we consider $\forall^2$ HyperLTL formulas. The proofs can be generalized. We denote $\mathcal{M}_\varphi[t/\pi]$ as the monitor where trace variable π of the template Monitor $\mathcal{M}_\varphi$ is initialized with explicit trace t.

Lemma 3. *Let φ be a $\forall^2$ HyperLTL formula over trace variables $\{\pi_1, \pi_2\}$. Given an arbitrary trace set T and an arbitrary trace t, $T \cup \{t\}$ is a model of φ if and only if T is still accepted by the following two monitors: (1) only π_1 is initialized with t (2) only π_2 is initialized with t. Formally, the following equivalence holds.*

$$\forall T \subseteq TR, \forall t \in TR.\, T \cup \{t\} \in \mathcal{L}(\varphi) \Leftrightarrow T \subseteq \mathcal{L}(\mathcal{M}_\varphi[t/\pi_1]) \wedge T \subseteq \mathcal{L}(\mathcal{M}_\varphi[t/\pi_2])$$

Lemma 4. *Given a $\forall^2$ HyperLTL formula φ over trace variables $\mathcal{V} := \{\pi_1, \ldots, \pi_n\}$ and two traces $t, t' \in TR$, the following holds: t dominates t' if and only if*

$$\mathcal{L}(\mathcal{M}_\varphi[t/\pi_1]) \subseteq \mathcal{L}(\mathcal{M}_\varphi[t'/\pi_1]) \wedge \mathcal{L}(\mathcal{M}_\varphi[t/\pi_2]) \subseteq \mathcal{L}(\mathcal{M}_\varphi[t'/\pi_2])$$

Proof. Assume for the sake of contradiction that (a) t dominates t' and w.l.o.g. (b) $\mathcal{L}(\mathcal{M}_\varphi[t/\pi_1]) \nsubseteq \mathcal{L}(\mathcal{M}_\varphi[t'/\pi_1])$. Thus, by definition of subset, there exists a trace $\tilde{t}$ with $\tilde{t} \in \mathcal{L}(\mathcal{M}_\varphi[t/\pi_1])$ and $\tilde{t} \notin \mathcal{L}(\mathcal{M}_\varphi[t'/\pi_1])$. Hence, $\Pi = \{\pi_1 \mapsto t, \pi_2 \mapsto \tilde{t}\}$ is a valid trace assignment, whereas $\Pi' = \{\pi_1 \mapsto t', \pi_2 \mapsto \tilde{t}\}$ is not. On the other hand, from (a) the following holds by Definition 5: $\forall T'$ with $\{t\} \subseteq T'$ it holds that $T' \in \mathcal{L}(\varphi) \Leftrightarrow T' \cup \{t'\} \in \mathcal{L}(\varphi)$. We choose T' as $\{t, \tilde{t}\}$, which is a contradiction to the equivalence since we know from (a) that Π is a valid trace assignment, but Π' is not a valid trace assignment.

For the other direction, assume that $\mathcal{L}(\mathcal{M}_\varphi[t/\pi_1]) \subseteq \mathcal{L}(\mathcal{M}_\varphi[t'/\pi_1])$ and $\mathcal{L}(\mathcal{M}_\varphi[t/\pi_2]) \subseteq \mathcal{L}(\mathcal{M}_\varphi[t'/\pi_2])$. Let T' be arbitrary such that $\{t\} \subseteq T'$. We distinguish two cases:

```
input  : $\forall^n$ HyperLTL formula $\varphi$,
         redundancy free set of traces $T$
         trace $t$
output: redundancy free set of traces $T_{min} \subseteq T \cup \{t\}$
$\mathcal{M}_\varphi = \texttt{build\_template}(\varphi)$
foreach $t' \in T$ do
    if $\bigwedge_{\pi \in \mathcal{V}} \mathcal{L}(\mathcal{M}_\varphi[t'/\pi]) \subseteq \mathcal{L}(\mathcal{M}_\varphi[t/\pi])$ then
        return $T$
    end
end
foreach $t' \in T$ do
    if $\bigwedge_{\pi \in \mathcal{V}} \mathcal{L}(\mathcal{M}_\varphi[t/\pi]) \subseteq \mathcal{L}(\mathcal{M}_\varphi[t'/\pi])$ then
        $T := T \setminus \{t'\}$
    end
end
return $T \cup \{t\}$
```

Fig. 3. Storage Minimization Algorithm.

– Case $T' \in \mathcal{L}(\varphi)$, then (a) $T' \subseteq \mathcal{L}(M_\varphi[t/\pi_1]) \subseteq \mathcal{L}(M_\varphi[t'/\pi_1])$ and (b) $T' \subseteq \mathcal{L}(M_\varphi[t/\pi_2]) \subseteq \mathcal{L}(M_\varphi[t'/\pi_2])$. By Lemma 3 and $T' \in \mathcal{L}(\varphi)$, it follows that $T' \cup \{t'\} \in \mathcal{L}(\varphi)$.
– Case $T' \notin \mathcal{L}(\varphi)$, then $T' \cup \{\hat{t}\} \notin \mathcal{L}(\varphi)$ for an arbitrary trace $\hat{t}$.

A generalization leads to the following theorem, which serves as the foundation of our trace storage minimization algorithm.

Theorem 6. *Given a $\forall^n$ HyperLTL formula φ over trace variables $\mathcal{V} := \{\pi_1, \ldots, \pi_n\}$ and two traces $t, t' \in TR$, the following holds: t dominates t' if and only if*

$$\bigwedge_{\pi \in \mathcal{V}} \mathcal{L}(\mathcal{M}_\varphi[t/\pi]) \subseteq \mathcal{L}(\mathcal{M}_\varphi[t'/\pi]) \ .$$

Corollary 2. *Given an $\exists^n$ HyperLTL formula φ over trace variables $\mathcal{V} := \{\pi_1, \ldots, \pi_n\}$ and two traces $t, t' \in TR$, the following holds: t dominates t' if and only if $\bigwedge_{\pi \in \mathcal{V}} \mathcal{L}(\mathcal{M}_\varphi[t'/\pi]) \subseteq \mathcal{L}(\mathcal{M}_\varphi[t/\pi])$.*

Theorem 7. *Algorithm 3 preserves the minimal trace set T, i.e., for all $t \in T$ it holds that t is not $(T \setminus \{t\}, \varphi)$-redundant.*

5 Monitoring Alternating HyperLTL Formulas

With the classic definition of monitorability (cf. Section 2), hardly any alternating HyperLTL formula is monitorable as their satisfaction cannot be characterized by a finite trace set, even for safety properties. Consider, for example, the formula $\varphi = \forall \pi. \exists \pi'. \square(a_\pi \rightarrow b_{\pi'})$. Assume a finite set of traces T does not violate the formula. Then, one can construct a new trace t where $a \in t[i]$ and

$b \notin t[i]$ for some position i, and for all traces $t' \in T$ it holds that $b \notin t'[i]$. Thus, the new trace set violates φ. Likewise, if there is a finite set of traces that violates φ, a sufficiently long trace containing only b's stops the violation.

If we fix a set of traces, we can check the satisfaction of an alternating HyperLTL formula with a modification of the offline monitoring algorithm presented earlier. This way, we can verify alternating hyperproperties after the execution of a system based on recorded traces. In our conference management system example, the property *"There was a submission for every paper that is visible for a program committee member."* is a hyperproperty that utilizes alternation and can be formalized as the $\forall\exists$HyperLTL formula

$$\forall \pi.\, \exists \pi'.\, pc_{\pi'} \wedge (\neg pc_{\pi} \rightarrow \bigcirc\Box(s_{\pi} \rightarrow \bigcirc v_{\pi'})) \ . \tag{14}$$

In the following, we present an extension to our offline algorithm for monitoring $\forall\exists$HyperLTL and $\exists\forall$HyperLTL formulas. Further, we show that the trace storage minimization technique is also applicable for alternating HyperLTL formulas, allowing to determine at runtime whether a trace needs to be stored or not.

We considered offline monitoring of universally quantified $\forall^n$HyperLTL in Section 3 by checking whether $\mathcal{M}_\varphi$ accepts N for every $N \in T^n$, given a trace set T and a HyperLTL formula φ. In contrast, an offline monitor for a $\forall^n\exists^m$HyperLTL and $\exists^m\forall^n$HyperLTL formula has to perform the checks

$$\bigwedge_{N \in T^n} \bigvee_{M \in T^m} \text{check if } \mathcal{M}_\varphi \text{ accepts } N \times M \ \text{, and}$$

$$\bigvee_{M \in T^m} \bigwedge_{N \in T^n} \text{check if } \mathcal{M}_\varphi \text{ accepts } M \times N \ \text{, respectively.}$$

We give a characterization of the trace dominance introduced in the last section for HyperLTL formulas with one alternation. These characterizations can be checked similarly to the algorithm depicted in Fig. 3.

Theorem 8. *Given a HyperLTL formula $\forall \pi.\, \exists \pi'.\, \psi$ two traces $t, t' \in TR$, the following holds: t dominates t' if and only if*

$$\mathcal{L}(\mathcal{M}_\varphi[t/\pi]) \subseteq \mathcal{L}(\mathcal{M}_\varphi[t'/\pi]) \text{ and } \mathcal{L}(\mathcal{M}_\varphi[t'/\pi']) \subseteq \mathcal{L}(\mathcal{M}_\varphi[t/\pi']) \ .$$

Corollary 3. *Given a HyperLTL formula $\exists \pi.\, \forall \pi'.\, \psi$ two traces $t, t' \in TR$, the following holds: t dominates t' if and only if*

$$\mathcal{L}(\mathcal{M}_\varphi[t'/\pi]) \subseteq \mathcal{L}(\mathcal{M}_\varphi[t/\pi]) \text{ and } \mathcal{L}(\mathcal{M}_\varphi[t/\pi']) \subseteq \mathcal{L}(\mathcal{M}_\varphi[t'/\pi']) \ .$$

Example 3. We show the effect of the dominance characterization on two example formulas. Consider the HyperLTL formula $\forall \pi.\, \exists \pi'.\, \Box(a_\pi \rightarrow b_{\pi'})$ and the traces $\{b\}\emptyset$, $\{b\}\{b\}$, $\{a\}\emptyset$, and $\{a\}\{a\}$. Trace $\{a\}\{a\}$ dominates trace $\{a\}\emptyset$ as instantiating π requires two consecutive b's for π' where $\{a\}\emptyset$ only requires a b at the first position (both traces do not contain b's, so instantiating π' leads to the same language). Similarly, one can verify that $\{b\}\{b\}$ dominates trace $\{b\}\emptyset$.

Consider alternatively the formula $\exists \pi.\, \forall \pi'.\, \Box(a_\pi \rightarrow b_{\pi'})$. In this case, $\{a\}\emptyset$ dominates $\{a\}\{a\}$ and $\{b\}\emptyset$ dominates $\{b\}\{b\}$.

Table 1. Specification Analysis for universally quantified hyperproperties.

		symm	trans	refl
ObsDet1	$\forall\pi.\forall\pi'.\ \square(I_\pi = I_{\pi'}) \rightarrow \square(O_\pi = O_{\pi'})$	✓	✗	✓
ObsDet2	$\forall\pi.\forall\pi'.\ (I_\pi = I_{\pi'}) \rightarrow \square(O_\pi = O_{\pi'})$	✓	✗	✓
ObsDet3	$\forall\pi.\forall\pi'.(O_\pi = O'_\pi)\ \mathcal{W}\ (I_\pi \neq I'_\pi)$	✓	✗	✓
QuantNoninf	$\forall\pi_0 \ldots \forall\pi_c.\ \neg((\bigwedge_i I_{\pi_i} = I_{\pi_0}) \wedge \bigwedge_{i \neq j} O_{\pi_i} \neq O_{\pi_j})$	✓	✗	✓
EQ	$\forall\pi.\forall\pi'.\square(a_\pi \leftrightarrow a_{\pi'})$	✓	✓	✓
ConfMan	$\forall\pi\forall\pi'.\ \big((\neg pc_\pi \wedge pc_{\pi'}) \rightarrow \bigcirc\square(s_\pi \rightarrow \bigcirc v_{\pi'})\big)$ $\wedge\big((pc_\pi \wedge pc_{\pi'}) \rightarrow \bigcirc\square(v_\pi \leftrightarrow v'_\pi)\big)$	✗	✗	✗

For our conference management example formula given in Equation 14, a trace $\{pc\}\emptyset\{v\}$ dominates $\{pc\}\emptyset\emptyset$ and $\emptyset\{s\}\emptyset$ dominates $\emptyset\emptyset\emptyset$, but $\emptyset\{s\}\emptyset$ and $\{pc\}\emptyset\{v\}$ are incomparable with respect to the dominance relation.

6 Evaluation

In this section, we report on experimental results of the presented algorithm and the accompanying optimizations. Our results show that those optimizations are orthogonal, i.e., none of the techniques subsumes the other. For the specification analysis, we checked variations of observational determinism, quantitative non-interference [17], equality and our conference management example for symmetry, transitivity, and reflexivity. The results are depicted in Table 1. The specification analysis comes with low costs (every check was done in under a second), but with a high reward in terms of constructed monitor instances (see Fig. 5). For hyperproperties that do not satisfy one of the properties, e.g., our conference management example, our trace analysis will still dramatically reduce the memory consumption.

For evaluating our trace analysis, we use a scalable, bounded variation of observational determinism: $\forall\pi.\forall\pi'.\square_{<n}(I_\pi = I_{\pi'}) \rightarrow \square_{<n+c}(O_\pi = O_{\pi'})$. Figure 4 shows a family of plots for this benchmark class, where c is fixed to three. We randomly generated a set of 10^5 traces. The blue (dashed) line depicts the number of traces that need to be stored, the red (dotted) line the number of traces that violated the property, and the green (solid) line depicts the pruned traces. When *increasing the requirements* on the system, i.e., decreasing n, we prune the majority of incoming traces with our trace analysis techniques.

In Fig. 5 we compare the running time of the monitoring optimizations presented in this paper to the naive approach. As a specification, we use the observational determinism property with a single input and a single output proposition. We compare the naive monitoring approach to the monitor using specification analysis and trace analysis, as well as a combination thereof. We randomly built traces of length 2000, with one byte of low input, i.e., one atomic proposition is allowed to appear for 8 steps. The remaining atomic propositions are one low output and five high in and outputs. Applying both of our techniques results in a tremendous speed up of the monitoring algorithm.

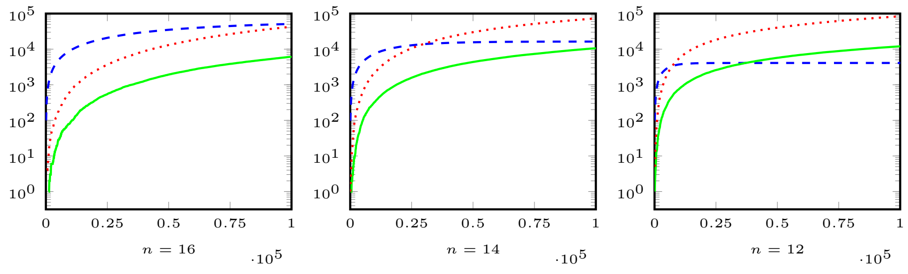

Fig. 4. Absolute numbers of violations in red (dotted), number of instances stored in blue (dashed), number of instances pruned in green (solid) for 10^5 randomly generated traces of length 100000. The y axis is scaled logarithmically.

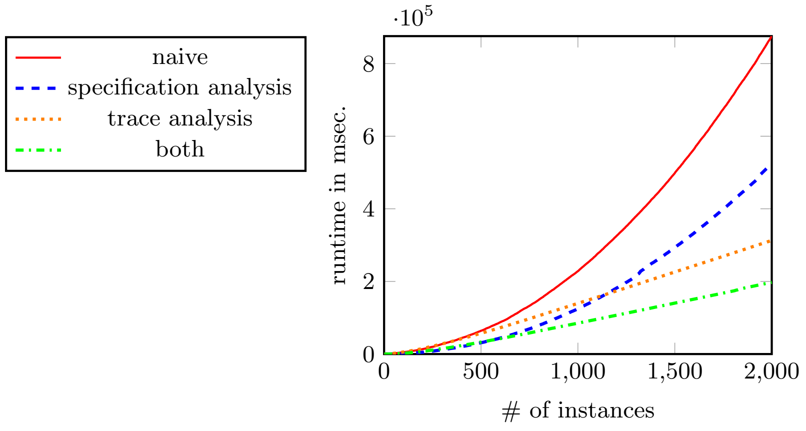

Fig. 5. Runtime comparison of naive monitoring approach with a version using specification analysis, trace analysis, and a combination of both.

7 Conclusion

In this paper, we have presented an automata based monitoring approach for HyperLTL. We showed that deciding whether an alternation-free formula is monitorable is PSPACE-complete. We presented two optimizations tackling different problems in monitoring hyperproperties. Trace analysis minimizes the needed memory, by minimizing the stored set of traces. Specification analysis reduces the algorithmic workload by reducing the number of comparisons between a newly observed trace and the previously stored traces. Combined, we have made significant progress towards the practical monitoring of hyperproperties.

References

1. Agrawal, S., Bonakdarpour, B.: Runtime verification of k-safety hyperproperties in HyperLTL. In: Proceedings of CSF. pp. 239–252. IEEE Computer Society (2016)

2. Askarov, A., Sabelfeld, A.: Tight enforcement of information-release policies for dynamic languages. In: Proceedings of CSF. pp. 43–59. IEEE Computer Society (2009)
3. Austin, T.H., Flanagan, C.: Permissive dynamic information flow analysis. In: Proceedings of PLAS. p. 3. ACM (2010)
4. Bauer, A.: Monitorability of omega-regular languages. CoRR abs/1006.3638 (2010)
5. Bichhawat, A., Rajani, V., Garg, D., Hammer, C.: Information flow control in webkit's javascript bytecode. In: Proceedings of POST. LNCS, vol. 8414, pp. 159–178. Springer (2014)
6. Bonakdarpour, B., Finkbeiner, B.: Runtime verification for HyperLTL. In: Proceedings of RV. LNCS, vol. 10012, pp. 41–45. Springer (2016)
7. Brett, N., Siddique, U., Bonakdarpour, B.: Rewriting-based runtime verification for alternation-free HyperLTL. In: Proceedings of TACAS. LNCS, vol. 10206, pp. 77–93 (2017)
8. Chudnov, A., Kuan, G., Naumann, D.A.: Information flow monitoring as abstract interpretation for relational logic. In: Proceedings of CSF. pp. 48–62. IEEE Computer Society (2014)
9. Clarkson, M.R., Finkbeiner, B., Koleini, M., Micinski, K.K., Rabe, M.N., Sánchez, C.: Temporal logics for hyperproperties. In: Proceedings of POST. LNCS, vol. 8414, pp. 265–284. Springer (2014)
10. Clarkson, M.R., Schneider, F.B.: Hyperproperties. Journal of Computer Security 18(6), 1157–1210 (2010)
11. d'Amorim, M., Rosu, G.: Efficient monitoring of omega-languages. In: Proceedings of CAV. LNCS, vol. 3576, pp. 364–378. Springer (2005)
12. Devriese, D., Piessens, F.: Noninterference through secure multi-execution. In: Proceedings of SP. pp. 109–124. IEEE Computer Society (2010)
13. Dimitrova, R., Finkbeiner, B., Kovács, M., Rabe, M.N., Seidl, H.: Model checking information flow in reactive systems. In: Proceedings of VMCAI. LNCS, vol. 7148, pp. 169–185. Springer (2012)
14. Dimitrova, R., Finkbeiner, B., Rabe, M.N.: Monitoring temporal information flow. In: Proceedings of ISoLA. LNCS, vol. 7609, pp. 342–357. Springer (2012)
15. Finkbeiner, B., Hahn, C.: Deciding hyperproperties. In: Proceedings of CONCUR. LIPIcs, vol. 59, pp. 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016)
16. Finkbeiner, B., Hahn, C., Stenger, M.: Eahyper: Satisfiability, implication, and equivalence checking of hyperproperties. In: Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II. pp. 564–570 (2017)
17. Finkbeiner, B., Rabe, M.N., Sánchez, C.: Algorithms for model checking HyperLTL and HyperCTL*. In: Proceedings of CAV. LNCS, vol. 9206, pp. 30–48. Springer (2015)
18. Guernic, G.L., Banerjee, A., Jensen, T.P., Schmidt, D.A.: Automata-based confidentiality monitoring. In: Proceedings of ASIAN. LNCS, vol. 4435, pp. 75–89. Springer (2006)
19. Kovács, M., Seidl, H.: Runtime enforcement of information flow security in tree manipulating processes. In: Proceedings of ESSoS. LNCS, vol. 7159, pp. 46–59. Springer (2012)
20. Manna, Z., Pnueli, A.: Temporal verification of reactive systems - safety. Springer (1995)
21. McLean, J.: Proving noninterference and functional correctness using traces. Journal of Computer Security 1(1), 37–58 (1992)

22. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977. pp. 46–57 (1977)
23. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Proceedings of FM. LNCS, vol. 4085, pp. 573–586. Springer (2006)
24. Roscoe, A.W.: CSP and determinism in security modelling. In: Proceedings of SP. pp. 114–127. IEEE Computer Society (1995)
25. Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications 21(1), 5–19 (2003)
26. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. In: Proceedings of STOC. pp. 159–168. ACM (1982)
27. Suh, G.E., Lee, J.W., Zhang, D., Devadas, S.: Secure program execution via dynamic information flow tracking. In: Proceedings of ASPLOS. pp. 85–96. ACM (2004)
28. Tabakov, D., Rozier, K.Y., Vardi, M.Y.: Optimized temporal monitors for systemc. Formal Methods in System Design 41(3), 236–268 (2012)
29. Vanhoef, M., Groef, W.D., Devriese, D., Piessens, F., Rezk, T.: Stateful declassification policies for event-driven programs. In: Proceedings of CSF. pp. 293–307. IEEE Computer Society (2014)
30. Zdancewic, S., Myers, A.C.: Observational determinism for concurrent program security. In: Proceedings of CSF. p. 29. IEEE Computer Society (2003)