Saarland University
Faculty of Mathematics and Computer Science

Master's Thesis

Satisfiability and Monitoring of Hyperproperties

Author:
Christopher Hahn

Supervisor:
Prof. Bernd Finkbeiner, Ph.D.

Reviewers:
Prof. Bernd Finkbeiner, Ph.D.
Prof. Dr. Gert Smolka

Submitted: 22nd August 2017

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath:

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent:

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, 22th August, 2017

Preface

The technical sections of Chapter 3 were published in "EAHyper: Satisfiability, Implication, and Equivalence Checking of Hyperproperties" [22] at the 29th International Conference on Computer Aided Verification (CAV 2017) and were presented by the author from July 24 - 28, 2017 in Heidelberg, Germany. The technical sections of Chapter 4 will be published as parts of the paper "Monitoring Hyperproperties" [23] at the 17th International Conference on Runtime Verification (RV2017) and will be presented by the author from September 13 - 16, 2017 in Seattle, USA.

Acknowledgements

I would like to express my deepest gratitude to my advisor and co-author Prof. Bernd Finkbeiner, Ph.D. His targeted guidance and outstanding support is more than I could have ever asked for. I look forward to the productive and compelling journey under his wings.

Also, I would like to thank Prof. Dr. Gert Smolka for reviewing this thesis. His acknowledgement of my scientific work means a lot to me, as he was one of the most influential lecturers during my undergraduate studies at Saarland University.

I thank the whole Reactive Systems Group at Saarland University for numerous, surprisingly productive, coffee breaks. Many thanks to my co-authors Marvin Stenger and Leander Tentrup for their hard work they put into our projects.

I thank my parents Vilma and Peter, my sister Nadine, my brothers Jonathan and Maximilian and, moreover, Jana and André.

Last but not least, I appreciate the enormous support of my beloved girlfriend Vivien. I cannot thank you enough.

Abstract

In this thesis, we revisit the satisfiability problem of hyperproperties from a practical point of view and study the runtime verification of hyperproperties. Hyperproperties, such as noninterference and observational determinism relate multiple execution traces with each other and are thus not expressible in standard temporal logics like LTL, CTL and CTL*. HyperLTL extends Linear-time Temporal Logic (LTL) with explicit trace quantification to express hyperproperties. We present the first practical satisfiability solver, called EAHyper, for hyperproperties expressed in the decidable fragment of HyperLTL. Applications of EAHyper include the automatic detection of specifications that are inconsistent or vacuously true, as well as the comparison of multiple formalizations of the same policy, such as different notions of observational determinism. Furthermore, we investigate the runtime verification problem of HyperLTL formulas. As hyperproperties relate multiple execution traces, 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. We present a technique that reduces the set of traces that new traces must be compared against to a minimal subset. We show that this leads to much more scalable monitoring with, in particular, significantly lower memory consumption. Additionally, we show the practical relevance of EAHyper in modern verification procedures. EAHyper can be used to avoid overhead during the monitoring process, by analyzing specifications given in HyperLTL.

Contents

Chapter 1

Introduction

In contrast to classic verification methods like model checking and theorem proving, which check whether an entire system is correct, **runtime verification** is concerned with the question whether a **run** of a system under consideration satisfies a given **formal specification** [29]. Temporal logics like Linear-time Temporal Logic (LTL) [35] are used as a well-studied rigorous formalism for specifying **trace properties**, such as mutual exclusion. In essence, runtime verification of trace properties is implemented by a **monitor** that performs the following membership test: $t \in \mathit{Traces}(\varphi)$, where t is an execution trace of the system, and $\mathit{Traces}(\varphi)$ is the trace property, i.e., a trace set given by an LTL formula φ. Runtime verification of trace properties on reactive systems, i.e., systems that continuously interact with an environment has been studied extensively in theory (e.g. [29, 20, 27, 5]) and practice (e.g. [37, 39, 33]). At first glance, general verification methods, such as model checking, seem to subsume runtime verification approaches, but this is, in fact, not the case in real-world systems that deal with uncertainty. For example, consider ships and airplanes where certain informations, such as a combination of weather, exact GPS position and many more environment variables, are only available during runtime or are simply infeasible to compute beforehand. Thus, monitoring safety-critical systems is a necessary verification layer to ensure that, in an unexpected case, either a human is alerted or an enforcement mechanism, which tries to stabilize the system again, is triggered.

However, there are properties which refer to multiple execution traces at the same time and are, thus, no trace properties, but **hyperproperties** [11]. The study of runtime verification of hyperproperties is fairly new and has recently received a lot of attention [1, 8, 7], due to the following practical motivation.

In contrast to safety critical systems, we are in practice often concerned with **privacy** critical systems, i.e., systems that contain sensitive data. For example, medical patient data and credit card informations should never leak into the public domain. Multiple **information flow policies** were designed to prevent such **information**

leakage. Analogously to the motivation for monitoring of safety-critical systems, we study the monitoring of privacy-critical systems to prevent information leakage in systems under uncertainty. Information flow policies are hyperproperties, since we consider multiple execution traces at a time. Thus, they are not expressible in standard verification languages like LTL, CTL and CTL*. For example, by distinguishing between two security levels (high-security and low-security), **observational determinism** [32, 38, 48] establishes the following information flow policy:

"The system behaves deterministically for a low-security observer."

Intuitively, observational determinism requires that the low security output, i.e., a publicly observable output, is the same on two executions of a system as long as the low-security input is the same on those executions as well. A system that satisfies observational determinism is therefore secure against the leakage of private information, since no altering of high security data is visible to an observer.

HyperLTL [12], which is an extension of LTL with quantification over traces, was introduced as a general specification language for expressing hyperproperties of practical interest. For example, we can express the above mentioned observational determinism in HyperLTL as follows:

$$\forall \pi. \forall \pi'. (\mathit{lowOut}_{\pi} \leftrightarrow \mathit{lowOut}_{\pi'}) \,\mathsf{W}\, (\mathit{lowIn}_{\pi} \not\leftrightarrow \mathit{lowIn}_{\pi'}).$$

We read this formula as "For every trace pair (π, π'), the low output on π is the same as the low output on π' as long as the low input is the same on π and π'". The temporal operator W is the weak version of the LTL until-operator U. By indexing atomic propositions with explicit trace variables, HyperLTL relates execution traces. Hence, HyperLTL is capable of expressing hyperproperties such as symmetry in mutual exclusion protocols [21] or information-flow policies, for example, observational determinism, generalized non-interference, and noninference [12].

As hyperproperties relate multiple traces with each other, runtime verification of hyperproperties is concerned with the question whether **runs** of a system under consideration satisfy a given hyperproperty. For this purpose, it is necessary to store previous seen traces in order to relate them to new traces. A naive monitor approach would therefore inevitable run out of memory quickly.

For the purpose of designing a more sophisticated algorithm that can be incorporated in a practical monitoring tool, we revisit the satisfiability problem of HyperLTL from a practical point of view. The idea is to use the decision procedure, given in [19], to build a powerful hyperproperty analyzation tool that can be used before verification processes to detect, for example, symmetry in HyperLTL formulas. We introduce the first practical HyperLTL-SAT solver **EAHyper** and explain a technique from the literature [23] in which our tool is efficiently used to speed up the monitoring procedure of HyperLTL formulas.

EAHyper implements the decision procedure for the $\exists^*\forall^*$ fragment, which is the largest decidable fragment. It contains in particular all alternation-free formulas and also all implications and equivalences between alternation-free formulas. EAHyper is a tool for the analysis of specifications given in HyperLTL and is especially useful for implication checking between different formalizations of information flow policies. For example, observational determinism can be formalized as the HyperLTL formula $\forall\pi.\forall\pi'.\ \square(\mathrm{I}_\pi = \mathrm{I}_{\pi'}) \rightarrow \square(\mathrm{O}_\pi = \mathrm{O}_{\pi'})$, or, alternatively, as the HyperLTL formula $\forall\pi.\forall\pi'.\ (\mathrm{O}_\pi = \mathrm{O}_{\pi'})\,\mathsf{W}\,(\mathrm{I}_\pi \neq \mathrm{I}_{\pi'})$. The first formalization states that on any pair of traces, where the inputs are the same, the outputs must be the same as well; the second formalization states that differences in the observable output may only occur **after** differences in the observable input have occurred. As can be easily checked with EAHyper, the second formalization is the stronger requirement. Furthermore, EAHyper can be used to identify properties of HyperLTL formulas. For example, both variations of observational determinism above are **symmetric** [23] and, hence, every symmetric monitor can be omitted.

We present a monitoring algorithm that, in addition to the optimization which uses EAHyper, comes with a technique to reduce the set of traces that new traces must be compared against to a minimal subset. Using our trace storage minimization technique results in significantly lower memory consumption.

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 ν 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 $\nu \in t'[i+1]$ for every $i \geqslant 0$. When T satisfies the property, $T \cup \{t^*\}$, where t^* is a new trace, amounts to checking every pair (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: it is inevitable that the monitoring problem becomes 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*	(1.1)
{}	{s}	{}	{}	{}	*an author submits a paper after one time unit*	(1.2)
{}	{s}	{s}	{}	{}	*an author submits two papers*	(1.3)

A satisfying program committee trace would be $\{pc\}\{\nu\}\{\nu\}\{\nu\}\emptyset$ as there are author traces with paper submissions at time step 0, 1, and 2. For checking our property, one can safely discard trace 1.2 as it poses no more requirements than trace 1.3. We say that trace 1.3 dominates trace 1.2. We show that, given a hyperproperty in 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.

Related Work. The model checking problem [21] as well as the satisfiability problem [25, 19] of HyperLTL were studied in the literature. Surprisingly with the following major results: for the quantifier alternation-free fragments of HyperLTL both problems remain as expensive as the corresponding problems for LTL, i.e., PSPACE-complete. MCHyper [21] was introduced as the first practical model checker for alternation-free HyperLTL formulas.

For SecLTL [15], which is subsumed by HyperLTL [12], a white box monitoring algorithm based on alternating automata was proposed [16]. In contrast, our approach has no access to the systems implementation (black box). The black box runtime verification problem of HyperLTL formulas was studied in the literature as well [1, 8, 7]. The authors present a quick overview over the monitoring problem of HyperLTL and its challenges in [7]. In [1], the authors present the first monitoring approach for HyperLTL based on petri nets and a syntactic classification of monitorable HyperLTL formulas. Their classification is extended in this thesis

to a complete procedure to decide whether a HyperLTL formula is monitorable. In [8], the authors present the first rewrite based monitoring algorithm of HyperLTL. Similar to our approach, they identify atomic propositions of interest with a book keeping function. However, our algorithm stores only the relevant traces and is therefore able to return a counter example. Additionally, in our approach, a HyperLTL-SAT solver can be used to speed up the monitoring process even further.

Techniques for the enforcement of information flow policies include tracking dependencies at the hardware level [45], language-based monitors [40, 2, 3, 47, 6], and abstraction-based dependency tracking [24, 28, 9]. Secure multi-execution [14] 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. Another mechanism to enforce information flow policies in programs was studied in [34]. It will be interesting to study the enforcement problem of HyperLTL in future work to check if results from the literature translate to the enforcement of HyperLTL formulas.

Structure of this thesis. In the next chapter, we provide the necessary preliminaries on hyperproperties and HyperLTL. In Chapter 3, we introduce EAHyper and evaluate the tool on five different benchmarks: (1) we check various formalizations of observational determinism for implication, (2) we use quantitative noninterference as a challenging benchmark by increasing the number of quantifiers, (3) we analyze the implication of various symmetry constrains derived from a model checking case study in the literature, (4) we check the implication between formalizations of error resistant code formulas, and (5) we evaluate EAHyper on randomly generated formulas. In Chapter 4, we study the monitoring of HyperLTL formulas. In Section 4.1, we define the notion of monitorability for hyperproperties. Furthermore, we show that deciding whether a HyperLTL formula is monitorable is PSpace-complete. In Section 4.2, we define the finite trace semantics of HyperLTL, before presenting our online and offline automata-based monitoring algorithms in Section 4.3. In the subsequent Section 4.4, we present our technique for minimizing the set of traces that our monitoring algorithm has to store. We evaluate our technique in Section 4.6. We consider the offline monitoring of HyperLTL formulas with alternation in Section 4.5, before we sketch briefly the idea of a further optimization presented in [23], which uses EAHyper to speed up the monitoring process even further. We present related work and conclude this thesis in Chapter 5.

Chapter 2

Preliminaries

In extension to well-studied **trace properties**, such as mutual exclusion and access control, a **hyperproperty** is a set of sets of infinite execution traces. In this chapter, we will give a brief introduction to the temporal logics of trace properties (linear-time temporal logic, LTL) and the temporal logic of hyperproperties (HyperLTL).

2.1 Trace Properties, Hyperproperties, and their Temporal Logics

Formally, let AP be a set of **atomic propositions**. A **trace** t is an infinite sequence over subsets of the atomic propositions. We define the set of traces $TR \coloneqq (2^{AP})^{\omega}$. A subset $T \subseteq TR$ is called a **trace property**. We use the following notation to manipulate traces: let $t \in TR$ be a trace and $i \in \mathbb{N}$ be a natural number. $t[i]$ denotes the i-th element of t. Therefore, $t[0]$ represents the starting element of the trace. Let $j \in \mathbb{N}$ and $j \geqslant i$. $t[i,j]$ denotes the sequence $t[i]\ t[i+1] \dots t[j-1]\ t[j]$. $t[i,\infty]$ denotes the infinite suffix of t starting at position i.

LTL Syntax. Linear-time Temporal Logic (LTL) [35] combines the usual boolean connectives with temporal modalities such as the **Next** operator $\bigcirc$ and the **Until** operator U. The syntax of LTL is given by the following grammar:

$$\varphi \ ::= \ p \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \varphi \,\mathsf{U}\, \varphi$$

where $p \in AP$ is an atomic proposition. $\bigcirc\varphi$ means that φ holds in the **next** position of a trace; $\varphi_1 \,\mathsf{U}\, \varphi_2$ means that φ_1 holds **until** φ_2 holds. There are several derived operators, such as $\diamondsuit\varphi \equiv \mathit{true}\,\mathsf{U}\,\varphi$, $\square\varphi \equiv \neg\diamondsuit\neg\varphi$, and $\varphi_1\ W\varphi_2 \equiv (\varphi_1 \,\mathsf{U}\, \varphi_2) \vee \square\varphi_1$. $\diamondsuit\varphi$ states that φ will **eventually** hold in the future and $\square\varphi$ states that φ holds **globally**; $\mathcal{W}$ is the **weak** version of the **until** operator.

LTL Semantics. Let $p \in AP$ and $t \in TR$. The semantics of an LTL formula is defined as the smallest relation $\vDash$ that satisfies the following conditions:

$$
\begin{array}{lll}
t \vDash p & \text{iff} & p \in t[0] \\
t \vDash \neg\psi & \text{iff} & t \nvDash \psi \\
t \vDash \psi_1 \vee \psi_2 & \text{iff} & t \vDash \psi_1 \text{ or } t \vDash \psi_2 \\
t \vDash \bigcirc \psi & \text{iff} & t[1,\infty] \vDash \psi \\
t \vDash \psi_1 \, \mathsf{U} \, \psi_2 & \text{iff} & \text{there exists } i \geqslant 0 : t[i,\infty] \vDash \psi_2 \\
 & & \text{and for all } 0 \leqslant j < i \text{ we have } t[j,\infty] \vDash \psi_1
\end{array}
$$

Formulas expressed in LTL, define a set of traces, i.e., a trace property. For example, consider an arbiter that grants two processes (p_1 and p_2) access to a shared resource. The following LTL formula expresses that the arbiter should never grant both processes access at the same time:

$$\square\neg(g_{p_1} \wedge g_{p_2})$$

In contrast $H \in 2^{TR}$, i.e. a set of sets of traces, is called a **hyperproperty** [11]. By lifting trace properties to sets of trace properties, we gain a lot of expressiveness. In particular, relations between execution traces can be expressed as a hyperproperty. For example, the following information-flow policy **observational determinism** [32, 38, 48] is a hyperproperty:

"Regardless of the high security input, the program appears to be deterministic to a low security observer."

HyperLTL Syntax. HyperLTL [12] extends LTL with trace variables and explicit trace quantification. Let $\mathcal{V}$ be an infinite supply of trace variables. The syntax of HyperLTL is given by the following grammar:

$$
\begin{aligned}
\psi &::= \exists\pi.\psi \mid \forall\pi.\psi \mid \varphi \\
\varphi &::= a_\pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \varphi \, \mathsf{U} \, \varphi
\end{aligned}
$$

where $a \in AP$ is an atomic proposition and $\pi \in \mathcal{V}$ is a trace variable. Note that atomic propositions are indexed by trace variables. The quantification over traces makes it possible to express properties like “on all traces ψ must hold”, which is expressed by $\forall\pi.\,\psi$. Dually, one can express that “there exists a trace such that ψ holds”, which is denoted by $\exists\pi.\,\psi$. The derived operators $\diamondsuit$, $\square$, and $\mathcal{W}$ are defined as for LTL.

As we will see in the semantics below, a HyperLTL formula defines a **hyperproperty**, i.e., a set of sets of traces. A set T of traces satisfies the hyperproperty if it is

an element of this set of sets. We can express our example of observational determinism in HyperLTL as follows:

$$\forall \pi_1 \forall \pi_2 . \, \square(I_\pi = I_{\pi'}) \rightarrow \square(O_\pi = O_{\pi'})$$

Where $X_\pi = X_{\pi'}$ denotes that traces π and π' are equal with respect to a set $X \subseteq AP$, i.e., $\bigwedge_{x \in X}(x_\pi \leftrightarrow x_{\pi'})$.

HyperLTL Semantics. Formally, the semantics of HyperLTL formulas is given with respect to a **trace assignment** Π from $\mathcal{V}$ to TR, i.e., a partial function mapping trace variables to actual traces. $\Pi[\pi \mapsto t]$ denotes that π is mapped to t, with everything else mapped according to Π. $\Pi[i, \infty]$ denotes the trace assignment that is equal to $\Pi(\pi)[i, \infty]$ for all π.

$\Pi \vDash_T \exists \pi . \psi$	iff	there exists $t \in T : \Pi[\pi \mapsto t] \vDash_T \psi$
$\Pi \vDash_T \forall \pi . \psi$	iff	for all $t \in T : \Pi[\pi \mapsto t] \vDash_T \psi$
$\Pi \vDash_T a_\pi$	iff	$a \in \Pi(\pi)[0]$
$\Pi \vDash_T \neg \psi$	iff	$\Pi \nvDash_T \psi$
$\Pi \vDash_T \psi_1 \vee \psi_2$	iff	$\Pi \vDash_T \psi_1$ or $\Pi \vDash_T \psi_2$
$\Pi \vDash_T \bigcirc \psi$	iff	$\Pi[1, \infty] \vDash_T \psi$
$\Pi \vDash_T \psi_1 \, U \, \psi_2$	iff	there exists $i \geqslant 0 : \Pi[i, \infty] \vDash_T \psi_2$
		and for all $0 \leqslant j < i$ we have $\Pi[j, \infty] \vDash_T \psi_1$

HyperLTL-SAT is the problem of deciding whether there exists a **non-empty** set of traces T such that $\Pi \vDash_T \psi$, where Π is the empty trace assignment and $\vDash_T$ is the smallest relation satisfying the conditions above. If $\vDash_T \psi$, we call T a model of ψ. 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 $\mathcal{V} = \{\pi_1, \ldots, \pi_k\}$ over alphabet Σ. We define $\Sigma_\mathcal{V}$ to be the alphabet where p_π is interpreted as an atomic proposition for every $p \in AP$ and $\pi \in \mathcal{V}$. We denote by $\vDash_{LTL}$ 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 $p_\pi \in s$.

Chapter 3

Satisfiability of Hyperproperties

The satisfiability problem of HyperLTL (HyperLTL-SAT) asks whether there exists a non-empty trace set that satisfies the HyperLTL formula under consideration. HyperLTL-SAT has been studied in previous work [19]. Compared to more prominent verification procedures such as model checking, the relevance of sophisticated algorithms for satisfiability checking of temporal formulas is often wrongfully overlooked. In this chapter, we show how the decision procedure for HyperLTL-SAT [19] can be used to implement a tool for the analysis of hyperproperties formalized in HyperLTL. It has been shown that the satisfiability problem of HyperLTL is undecidable in general.

Theorem 3.1 *[19] HyperLTL-SAT is undecidable in general.*

However, a particular interesting fragment of HyperLTL is the **alternation-free** fragment. We call a HyperLTL formula φ (quantifier) **alternation-free** if and only if the quantifier prefix only consists of either only universal or only existential quantifiers. We denote the corresponding fragments as the $\forall^*$ and $\exists^*$ fragments, respectively. The alternation-free fragment of HyperLTL, since this fragment combines the best from two worlds: (1) the expressiveness of HyperLTL, since many practical relevant information-flow policies belong to this fragment and (2) the existence of efficient algorithms for model checking, satisfiability checking, and, as we will see in Section 4.1, for monitoring. Surprisingly, the satisfiability problem of alternation-free HyperLTL remains as expensive as LTL satisfiability[43].

Theorem 3.2 *[19] HyperLTL-SAT is* PSPACE*-complete for the alternation-free fragment.*

The $\exists^*\forall^*$ fragment is the largest decidable fragment. The $\exists^*\forall^*$ fragment consists of all HyperLTL formulas with at most one quantifier alternation, where no existential quantifier is in the scope of a universal quantifier. It contains in particular all alternation-free formulas and also all implications and equivalences between alternation-free formulas.

Theorem 3.3 *[19]* $\exists^*\forall^*$ *HyperLTL-SAT is* EXPSPACE*-complete.*

In the following section, we introduce EAHyper, the first HyperLTL-SAT solver. EAHyper implements the decision procedure for the $\exists^*\forall^*$ fragment of HyperLTL [19].

3.1 EAHyper - The First HyperLTL-SAT Solver

The input of EAHyper is either a HyperLTL formula in the $\exists^*\forall^*$ fragment, or an implication between two alternation-free formulas. For $\exists^*\forall^*$ formulas, EAHyper reports satisfiability; for implications between alternation-free formulas, validity. EAHyper proceeds in three steps:

1. **Translation into the $\exists^*\forall^*$ fragment:** If the input is an implication between two alternation-free formulas, we construct a formula in the $\exists^*\forall^*$ fragment that represents the **negation** of the implication. For example, for the implication of $\forall\pi_1 \dots \forall\pi_n.\psi$ and $\forall\pi'_1 \dots \forall\pi'_m.\varphi$, we construct the $\exists^*\forall^*$ formula $\exists\pi'_1 \dots \exists\pi'_m \forall\pi_1 \dots \forall\pi_n.\psi \wedge \neg\varphi$. The implication is valid if and only if the resulting $\exists^*\forall^*$ formula is unsatisfiable.

2. **Reduction to LTL satisfiability:** EAHyper implements the decision procedure for the $\exists^*\forall^*$ fragment of HyperLTL [19]. The satisfiability of the HyperLTL formula is reduced to the satisfiability of an LTL formula:

   - Formulas in the $\forall^*$ fragment are translated to LTL formulas by discarding the quantifier prefix and all trace variables. For example, $\forall\pi_1.\forall\pi_2.\square b_{\pi_1} \wedge \square\neg b_{\pi_2}$ is translated to the equisatisfiable LTL formula $\square b \wedge \square\neg b$.

   - Formulas in the $\exists^*$ fragment are translated to LTL formulas by introducing a fresh atomic proposition a_i for every atomic proposition a and every trace variable π_i. For example, $\exists\pi_1.\exists\pi_2.\ a_{\pi_1} \wedge \square\neg b_{\pi_1} \wedge \square b_{\pi_2}$ is translated to the equisatisfiable LTL formula $a_1 \wedge \square\neg b_1 \wedge \square b_2$.

   - Formulas in the $\exists^*\forall^*$ fragment are translated into the $\exists^*$ fragment (and then on into LTL) by unrolling the universal quantifiers. For example, $\exists\pi_1.\exists\pi_2.\forall\pi'_1.\forall\pi'_2.\square a_{\pi'_1} \wedge \square b_{\pi'_2} \wedge \square c_{\pi_1} \wedge \square d_{\pi_2}$ is translated to the equisatisfiable $\exists^*$ formula $\exists\pi_1.\exists\pi_2.(\square a_{\pi_1} \wedge \square b_{\pi_1} \wedge \square c_{\pi_1} \wedge \square d_{\pi_2}) \wedge (\square a_{\pi_2} \wedge \square b_{\pi_1} \wedge \square c_{\pi_1} \wedge \square d_{\pi_2}) \wedge (\square a_{\pi_1} \wedge \square b_{\pi_2} \wedge \square c_{\pi_1} \wedge \square d_{\pi_2}) \wedge (\square a_{\pi_2} \wedge \square b_{\pi_2} \wedge \square c_{\pi_1} \wedge \square d_{\pi_2})$.

3. **LTL satisfiability:** The satisfiability of the resulting LTL formula is checked through an external tool. Currently, EAHyper is linked to two LTL satisfiability checkers, pltl and Aalta.

Table 3.1: Quantitative noninterference benchmark: wall clock time in seconds for checking whether QN(row) implies QN(column). "–" denotes that the instance was not solved in 120 seconds.

(a) Aalta

QN	1	2	3	4	5
1	0.04	0.04	0.54	–	–
2	0.03	0.09	1.58	–	–
3	0.03	0.05	0.68	–	–
4	0.03	0.11	0.34	8.68	–
5	0.06	0.34	–	–	–

(b) pltl

QN	1	2	3	4	5
1	0.05	0.05	0.08	0.13	0.23
2	0.05	0.11	0.25	0.39	0.79
3	0.07	0.25	0.77	2.02	5.12
4	0.16	0.73	3.12	17.73	43.26
5	0.26	2.57	15.67	71.82	–

- Pltl [41] is a one-pass tableaux-based decision procedure for LTL, which not necessarily explores the full tableaux.
- Aalta_2.0 [30] is a decision procedure for LTL based on a reduction to the Boolean satisfiability problem, which is in turn solved by minisat [18]. Aalta's on-the-fly approach is based on so-called obligation sets and outperforms model-checking-based LTL satisfiability solvers.

EAHyper is implemented in OCaml and supports UNIX-based operating systems. Batch-processing of HyperLTL formulas is provided. Options such as the choice of the LTL satisfiability checker are provided via a command-line interface.

3.2 Experimental Results

We report on the performance of EAHyper on a range of benchmarks, including observational determinism, symmetry, error resistant code, as well as randomly generated formulas. The experiments were carried out in a virtual machine running Ubuntu 14.04 LTS on an Intel Core i5-2500K CPU with 3.3 GHZ and 2 GB RAM. We chose to run EAHyper in a virtual machine to make our results easily reproducible; running EAHyper natively results in (even) better performance.[1]

- **Observational Determinism [32, 38, 48].** Our first benchmark compares the following formalizations of observational determinism, with $|I| = |O| = 1$: $(OD1) : \forall\pi_1.\forall\pi_1'.\square(I_{\pi_1} = I_{\pi_1'}) \rightarrow \square(O_{\pi_1} = O_{\pi_1'})$, $(OD2) : \forall\pi_2.\forall\pi_2'.(I_{\pi_2} = I_{\pi_2'}) \rightarrow \square(O_{\pi_2} = O_{\pi_2'})$, and $(OD3) : \forall\pi_3.\forall\pi_3'.(O_{\pi_3} = O_{\pi_3'})\,W\,(I_{\pi_3} \neq I_{\pi_3'})$. EAHyper needs less then a second to order the formalizations with respect to implication: $OD2 \rightarrow OD1$, $OD2 \rightarrow OD3$, and $OD3 \rightarrow OD1$.

[1] EAHyper is available online at https://react.uni-saarland.de/tools/eahyper/.

Table 3.2: Error resistant codes benchmark: wall clock time in seconds for checking whether Ham(row) implies Ham(column).

Ham	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0.03	0.02	0.03	0.02	0.02	0.02	0.03	0.03	0.04	0.08	0.10	0.18	0.25	0.46	0.74	1.35	2.62
1	0.03	0.02	0.03	0.03	0.04	0.03	0.05	0.04	0.06	0.08	0.13	0.21	0.40	0.49	0.82	1.50	2.99
2	0.01	0.03	0.03	0.03	0.04	0.02	0.03	0.04	0.04	0.07	0.12	0.21	0.36	0.55	0.88	1.59	3.09
3	0.03	0.04	0.04	0.05	0.04	0.04	0.03	0.04	0.05	0.07	0.12	0.23	0.36	0.52	0.87	1.56	3.12
4	0.04	0.04	0.04	0.06	0.10	0.02	0.03	0.05	0.08	0.08	0.16	0.21	0.36	0.52	0.86	1.66	3.05
5	0.03	0.03	0.05	0.07	0.07	0.19	0.14	0.17	0.05	0.08	0.14	0.22	0.30	0.52	0.92	1.55	2.99
6	0.03	0.04	0.05	0.06	0.09	0.22	0.35	0.21	0.25	0.11	0.25	0.26	0.36	0.53	0.87	1.57	3.00
7	0.04	0.05	0.05	0.05	0.14	0.24	0.32	0.37	0.38	0.42	0.14	0.20	0.37	0.52	0.89	1.65	3.05
8	0.05	0.05	0.07	0.10	0.17	0.23	0.26	0.36	0.50	0.56	0.47	0.40	0.53	0.53	1.13	1.61	3.18
9	0.07	0.08	0.08	0.10	0.16	0.19	0.21	0.43	0.70	0.64	0.48	0.52	0.90	0.65	1.03	1.71	3.08
10	0.09	0.13	0.15	0.15	0.21	0.20	0.34	0.43	0.54	0.76	1.38	1.55	0.61	0.89	1.03	1.78	3.22
11	0.16	0.23	0.22	0.24	0.24	0.26	0.41	0.53	0.62	0.81	1.30	1.29	1.81	1.05	1.86	2.33	3.17
12	0.27	0.30	0.36	0.30	0.32	0.41	0.45	0.46	0.85	0.91	1.69	1.28	2.81	2.82	1.14	3.91	4.49
13	0.38	0.46	0.51	0.47	0.57	0.52	0.57	0.86	1.03	1.27	1.47	2.16	3.19	8.22	5.48	8.64	7.08
14	0.69	0.87	0.91	0.84	0.84	0.98	0.94	1.02	1.46	1.30	2.01	3.82	3.96	6.35	7.50	9.06	11.11
15	1.22	1.52	1.58	1.70	1.69	1.65	1.67	1.74	1.87	2.73	3.02	3.08	5.87	7.25	13.04	34.17	12.26
16	2.26	3.04	2.97	3.00	3.10	3.11	3.35	3.29	3.57	4.17	3.76	5.78	7.45	17.31	17.75	31.51	48.09

- **Quantitative Noninterference [10].** The bounding problem of quantitative noninterference asks whether the amount of information leaked by a system is bounded by a constant c. This is expressed in HyperLTL as the requirement that there are no $c+1$ distinguishable traces for a **low-security** observer [44].

$$QN(c) \coloneqq \forall \pi_0 \ldots \forall \pi_c. \neg((\bigwedge_i I_{\pi_i} = I_{\pi_0}) \wedge \bigwedge_{i \neq j} O_{\pi_i} \neq O_{\pi_j})$$

  In the benchmark, we check implications between different bounds. The performance of EAHyper is shown in Table 3.1. Using Aalta as the LTL satisfiability checker generally produces faster results, but pltl scales to larger bounds.

- **Symmetry [21].** A violation of symmetry in a mutual exclusion protocol indicates that some concurrent process has an unfair advantage in accessing a critical section. The benchmark is derived from a model checking case study, in which various symmetry claims were verified and falsified for the Bakery protocol. EAHyper checks the implications between the four main symmetry properties from the case study in 13.86 seconds. Exactly one of the implications turns out to be true.

- **Error resistant code [21, 26].** Error resistant codes enable the transmission of data over noisy channels. A typical model of errors bounds the number of flipped bits that may happen for a given code word length. Then, error correction coding schemes must guarantee that all code words have a minimal Hamming distance. The following HyperLTL formula specifies that all code words $o \in O$ produced by an encoder have a minimal Hamming distance [26]

Table 3.3: Random formulas benchmark: instances solved in 120 seconds and average wall clock time in seconds for 250 random formulas. Size denotes the tree-size argument for randltl.

size	40	60	40	60	40	60	40	60	40	60	40	60	40	60	40	60
	$\exists^0\forall^0$		$\exists^1\forall^0$		$\exists^2\forall^0$		$\exists^3\forall^0$		$\exists^4\forall^0$		$\exists^5\forall^0$		$\exists^6\forall^0$		$\exists^7\forall^0$		$\exists^8\forall^0$	
solved			250	250	250	250	250	250	250	250	250	250	250	250	250	250	250	250
avgt			0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
	$\exists^0\forall^1$		$\exists^1\forall^1$		$\exists^2\forall^1$		$\exists^3\forall^1$		$\exists^4\forall^1$		$\exists^5\forall^1$		$\exists^6\forall^1$		$\exists^7\forall^1$		$\exists^8\forall^1$	
solved	250	250	250	250	250	250	250	249	250	250	249	247	250	248	249	247	247	248
avgt	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.05	0.02	0.06	0.02	0.01	0.02	0.01	0.13	0.02	0.04	0.08
	$\exists^0\forall^2$		$\exists^1\forall^2$		$\exists^2\forall^2$		$\exists^3\forall^2$		$\exists^4\forall^2$		$\exists^5\forall^2$		$\exists^6\forall^2$		$\exists^7\forall^2$		$\exists^8\forall^2$	
solved	250	250	250	250	248	249	249	247	247	247	248	246	246	246	244	246	244	247
avgt	0.01	0.01	0.01	0.01	0.03	0.12	0.03	0.01	0.26	0.02	0.32	0.02	0.09	0.02	0.02	0.02	0.05	0.03
	$\exists^0\forall^3$		$\exists^1\forall^3$		$\exists^2\forall^3$		$\exists^3\forall^3$		$\exists^4\forall^3$		$\exists^5\forall^3$		$\exists^6\forall^3$		$\exists^7\forall^3$		$\exists^8\forall^3$	
solved	250	250	250	250	249	247	248	246	247	245	245	246	245	246	244	247	243	246
avgt	0.01	0.01	0.01	0.01	0.03	0.02	0.07	0.02	0.06	0.03	0.14	0.05	0.17	0.08	0.23	0.16	0.45	0.25
	$\exists^0\forall^4$		$\exists^1\forall^4$		$\exists^2\forall^4$		$\exists^3\forall^4$		$\exists^4\forall^4$		$\exists^5\forall^4$		$\exists^6\forall^4$		$\exists^7\forall^4$		$\exists^8\forall^4$	
solved	250	250	250	250	250	246	247	246	245	246	244	247	245	247	244	245	0	0
avgt	0.01	0.1	0.01	0.01	0.02	0.01	0.21	0.03	0.35	0.09	0.23	0.28	0.46	1.01	0.98	2.41	–	–
	$\exists^0\forall^5$		$\exists^1\forall^5$		$\exists^2\forall^5$		$\exists^3\forall^5$		$\exists^4\forall^5$		$\exists^5\forall^5$		$\exists^6\forall^5$		$\exists^7\forall^5$		$\exists^8\forall^5$	
solved	250	250	250	250	249	247	248	247	243	245	245	246	0	0	0	0	0	0
avgt	0.01	0.01	0.01	0.01	0.26	0.02	0.18	0.07	0.27	0.37	0.51	2.81	–	–	–	–	–	–

of d: $\forall\pi.\forall\pi'.\mathsf{F}(\bigvee_{i\in I}\neg(i_\pi \leftrightarrow i_{\pi'})) \rightarrow \neg \mathit{Ham}_O(d-1,\pi,\pi')$. Ham_O is recursively defined as $\mathit{Ham}_O(-1,\pi,\pi') = \mathit{false}$ and

$$\mathit{Ham}_O(d,\pi,\pi') = (\bigwedge_{o\in O} o_\pi \leftrightarrow o_{\pi'})\,\mathsf{W}\,(\bigvee_{o\in O}\neg(o_\pi \leftrightarrow o_{\pi'}) \wedge \bigcirc \mathit{Ham}_O(d-1,\pi,\pi')).$$

The benchmark checks implications between the HyperLTL formulas for different minimal Hamming distances. The performance of EAHyper is shown in Table 3.2.

- **Random formulas.** In the last benchmark, we randomly generated sets of 250 HyperLTL formulas containing five atomic propositions, using randltl [17] and assigning trace variables randomly to atomic propositions. As shown in Table 3.3, EAHyper reaches its limits, by running out of memory, after approximately five existential and five universal quantifiers. However, EAHyper shows encouraging results on the practical relevant fragments, such as the alternation-free fragment and the fragment of implication between information-flow policies ($\forall^2\exists^2$).

In this section, we presented EAHyper, which is the first satisfiability solver for hyperproperties formalized in the $\exists^*\forall^*$ fragment of HyperLTL. We evaluated our

implementation on various benchmark. The code of EAHyper as well as every benchmark presented in this thesis is available at Github [2]. Furthermore, we invite the interested reader to try out EAHyper in our online interface [3].

[2]https://github.com/reactive-systems/eahyper
[3]https://www.react.uni-saarland.de/tools/online/EAHyper/

Chapter 4

Monitoring of Hyperproperties

As hyperproperties relate multiple traces with each other, runtime verification of hyperproperties is concerned with the question whether **runs** of a system under consideration satisfy a given hyperproperty. Thus, there are many obstacles to overcome in monitoring hyperproperties (see [7] 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.

4.1 Monitorability

In 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]. 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., $U \preceq V \coloneqq \forall u \in U.\, \exists v \in V.\, u \preceq v$ 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.

Lemma 4.1 *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.*

Proof Assume that there is a trace assignment A over trace variables $\mathcal{V}$ such that $A \vDash_\emptyset \psi$. We define $w \subseteq \Sigma_\mathcal{V}^\omega$ such that $x_\pi \in w[i]$ if, and only if, $x \in A(\pi)[i]$ for all $i \geqslant 0$, $x \in AP$, and $\pi \in \mathcal{V}$. An induction over ψ shows that $w \vDash_{LTL} \psi$.

Assume ψ is satisfiable for $\vDash_{LTL}$, i.e., there exists a $w \subseteq \Sigma_\mathcal{V}^\omega$, such that $w \vDash_{LTL} \psi$. We construct an assignment A in the following manner: Let $\pi \in \mathcal{V}$ be arbitrary.

We map π to the trace t obtained by projecting the corresponding $p_\pi \in \Sigma_\mathcal{V}$, i.e., $\forall i \geqslant 0.\, t[i] = \#_\pi(w[i])$. ■

For ω-regular languages, monitorability is the property whether language containment can be decided by finite prefixes [36]. Given a ω-regular 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 4.2 *Given a HyperLTL formula $\varphi = \forall \pi_1 \ldots \forall \pi_k.\, \psi$, where ψ is an LTL formula. It holds that $\mathit{good}(\mathcal{L}(\varphi)) = \emptyset$ unless $\psi \equiv \mathit{true}$.*

Proof If $\psi \equiv \mathit{true}$ then $\mathcal{L}(\varphi) = \mathcal{P}(\Sigma^\omega)$ and $\mathit{good}(\mathcal{L}(\varphi)) = \mathcal{P}^*(\Sigma^*)$. Assume for contradiction that $\psi \not\equiv \mathit{true}$ and $\mathit{good}(\mathcal{L}(\varphi)) \neq \emptyset$, i.e., there is a finite set $U \subseteq \Sigma^*$ that is a good prefix set of φ. Since $\psi \neq \mathit{true}$, there is at least one infinite trace σ with $\sigma \not\models \psi$. We translate this trace to a set of infinite traces W where $W \not\models \varphi$ using Lemma 4.1. Further, for all $V \in \mathcal{P}(\Sigma^*)$ with $U \preceq V$, it holds that $W \subseteq V$, hence, $V \notin \mathcal{L}(\varphi)$ violating the assumption that $U \in \mathit{good}(\mathcal{L}(\varphi))$. ■

Theorem 4.3 *Given a HyperLTL formula $\varphi = \forall \pi_1 \ldots \forall \pi_k.\, \psi$, 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 $V \preceq W$, $W \not\models \varphi$, 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 4.2 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 $u \in \Sigma_V^*$, we translate it into the (canonical) U' and get a V' satisfying the conditions above. Let $v' \in \Sigma_V^*$ 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, $u'v' \in \mathit{bad}(\mathcal{L}(\psi))$. ∎

Corollary 4.4 *Given a HyperLTL formula $\varphi = \exists \pi_1 \dots \exists \pi_k.\psi$, where ψ is an LTL formula. φ is monitorable if, and only if,*

$$\forall u \in \Sigma_V^*.\exists v \in \Sigma_V^*.uv \in \mathit{good}(\mathcal{L}(\psi))$$

.

Theorem 4.5 *Given an alternation-free HyperLTL formula φ. Deciding whether φ is monitorable is* PSpace*-complete.*

Proof We consider the case that $\varphi = \forall \pi_1 \dots \forall \pi_2.\psi$, the case for existentially quantified formulas is dual. We apply the characterization from Theorem 4.3. First, we have to check validity of ψ which can be done in polynomial space [42]. Next, we have to determine whether $\forall u \in \Sigma_V^*.\exists v \in \Sigma_V^*.uv \in \mathit{bad}(\mathcal{L}(\psi))$. We use a slight modification of the PSpace algorithm given by Bauer [4]. Hardness follows as the problem is already PSpace-hard for LTL. ∎

4.2 Finite Trace Semantics.

We define a finite trace semantics for HyperLTL based on the finite trace semantics of LTL [31]. 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, $t[i,\dots]$ 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. $\Pi_{\mathit{fin}}[i,\dots]$ denotes the trace assignment that is equal to $\Pi_{\mathit{fin}}(\pi)[i,\dots]$ for all π. We define a subsequence of t as follows.

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

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

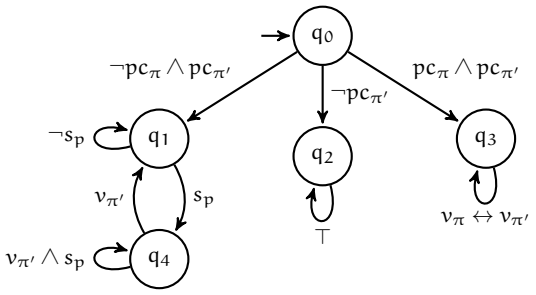

Figure 4.1: Visualization of a monitor template corresponding to formula given in Equation 4.1. We use a symbolic representation of the transition function δ.

4.3 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 $\Sigma = 2^{AP \times \mathcal{V}}$, 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 $(2^{AP})^*$, thus we define a run with respect to a n-ary tuple $N \in ((2^{AP})^*)^n$ 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 $0 \leqslant i < m$ it holds that

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

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 [13, 46].

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. 4.1.

$$\forall \pi . \forall \pi' . \left((\neg pc_\pi \wedge pc_{\pi'}) \to \bigcirc \Box (s_\pi \to \bigcirc v_{\pi'})\right) \wedge \left((pc_\pi \wedge pc_{\pi'}) \to \bigcirc \Box (v_\pi \leftrightarrow v'_\pi)\right) \quad (4.1)$$

The offline and online algorithms for monitoring HyperLTL formulas are presented

```
input : ∀^π HyperLTL formula φ
        set of traces T
output: satisfied or π-ary tuple
        witnessing violation
M_φ = build_template(φ);
for each tuple N ∈ T^π do
    if M_φ accepts N then
        proceed;
    else
        return N;
    end
end
return satisfied;
```

Algorithm 1: Offline Algorithm.

in Algorithm 1 and Algorithm 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 π-ary tuple $N \in T^\pi$. 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 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 $N \in (T \cup \{t\})^\pi$ containing the new trace t, i.e., $N \notin T^\pi$.

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.

4.4 Minimizing Trace Storage

The main obstacle in monitoring hyperproperties is the potentially unbounded space consumption. In the first section of this chapter, we present an analysis phase of our algorithm that 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.

The main idea of the trace analysis, considered in the following, is to check whether

input : $\forall^n$ HyperLTL formula φ
output: satisfied or n-ary tuple witnessing violation

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

Algorithm 2: Online Algorithm.

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 4.6 *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.*

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

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 4.1. Assume our algorithm observes the following three traces of length five.

{}	{s}	{}	{}	{}	*an author immediately submits a paper*	(4.2)
{}	{}	{s}	{}	{}	*an author submits a paper after one time unit*	(4.3)
{}	{}	{s}	{s}	{}	*an author submits two papers*	(4.4)

Trace 4.3 contains, with respect to φ above, no more information than trace 4.4. We say that trace 4.4 dominates trace 4.3 and, hence, trace 4.3 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 immediately submits a paper* (4.5)
{}	{}	{s}	{s}	{}	*an author submits two papers* (4.6)
{}	{pc}	{v}	{v}	{v}	*a PC member observes three papers* (4.7)

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 4.7 dominates trace 4.5 and trace 4.6, since the information that three papers have been submitted is preserved in trace 4.7. Hence, it suffices to remember the last trace to detect, for example, the following violations.

{}	{pc}	{v}	{v}	{v}	*a PC member observes three papers* (4.8)
{}	{pc}	{v}	{v}	{}	*↯a PC member observes two papers ↯* (4.9)

or

{}	{}	{}	{}	{s}	*↯an author submits a paper after three time units ↯* (4.10)

Note that none of the previous user traces, i.e., trace 4.2 to trace 4.6, are needed to detect a violation.

Definition 4.7 *Given* $t, t' \in TR$*, we say* t ***dominates*** t' *if* t' *is* $(\{t\}, \varphi)$*-redundant.*

The observations from the example above can be generalized to a language inclusion check (cf. Theorem 4.11), 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 4.8 *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 4.9 *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) $\mathcal{L}(\mathcal{M}_\varphi[t/\pi_1]) \not\subseteq \mathcal{L}(\mathcal{M}_\varphi[t'/\pi_1])$. 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 4.7: $\forall T'$ 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:

- 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 4.8 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 $\hat{t}$. ∎

Lemma 4.10 *Given an $\exists^2$ 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) $\mathcal{L}(\mathcal{M}_\varphi[t'/\pi_1]) \not\subseteq \mathcal{L}(\mathcal{M}_\varphi[t/\pi_1])$. 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 4.7: $\forall T'$ 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:

- Case $T' \cup \{t'\} \in \mathcal{L}(\varphi)$, 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 4.8 and $T' \cup \{t'\} \in \mathcal{L}(\varphi)$, it follows that $T' \in \mathcal{L}(\varphi)$.
- Case $T' \cup \{t'\} \notin \mathcal{L}(\varphi)$, then $T' \notin \mathcal{L}(\varphi)$. ∎

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

input : $\forall^n$ HyperLTL formula φ,
redundancy free set of traces T
trace t
output: redundancy free set of traces $\mathsf{T}_{min} \subseteq \mathsf{T} \cup \{\mathrm{t}\}$

$\mathcal{M}_\varphi$ = `build_template(`φ`)`

foreach $\mathrm{t}' \in \mathsf{T}$ **do**
  if $\bigwedge_{\pi \in \mathcal{V}} \mathcal{L}(\mathcal{M}_\varphi[\mathrm{t}'/\pi]) \subseteq \mathcal{L}(\mathcal{M}_\varphi[\mathrm{t}/\pi])$ **then**
    return T
  end
end
foreach $\mathrm{t}' \in \mathsf{T}$ **do**
  if $\bigwedge_{\pi \in \mathcal{V}} \mathcal{L}(\mathcal{M}_\varphi[\mathrm{t}/\pi]) \subseteq \mathcal{L}(\mathcal{M}_\varphi[\mathrm{t}'/\pi])$ **then**
    $\mathsf{T} := \mathsf{T} \setminus \{\mathrm{t}'\}$
  end
end
return $\mathsf{T} \cup \{\mathrm{t}\}$

Figure 4.2: Storage Minimization Algorithm.

Theorem 4.11 *Given a $\forall^n$ HyperLTL formula φ over trace variables $\mathcal{V} := \{\pi_1, \ldots, \pi_n\}$ and two traces $\mathrm{t}, \mathrm{t}' \in TR$, the following holds: t dominates t' if and only if*

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

Corollary 4.12 *Given an $\exists^n$ HyperLTL formula φ over trace variables $\mathcal{V} := \{\pi_1, \ldots, \pi_n\}$ and two traces $\mathrm{t}, \mathrm{t}' \in TR$, the following holds: t dominates t' if and only if $\bigwedge_{\pi \in \mathcal{V}} \mathcal{L}(\mathcal{M}_\varphi[\mathrm{t}'/\pi]) \subseteq \mathcal{L}(\mathcal{M}_\varphi[\mathrm{t}/\pi])$.*

Theorem 4.13 *Algorithm 3 preserves the minimal trace set T, i.e., for all $\mathrm{t} \in \mathsf{T}$ it holds that t is not $(\mathsf{T} \setminus \{\mathrm{t}\}, \varphi)$-redundant.*

Proof By induction on $\mathsf{T} \setminus \{\mathrm{t}\}$ and Theorem 4.11. ∎

4.5 Monitoring Alternating HyperLTL Formulas

In this chapter, we consider HyperLTL Formulas which are not alternation-free. With the classic definition of monitorability (cf. Section 4.1), 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 $\varphi = \forall\pi.\, \exists\pi'.\, \square(\mathrm{a}_\pi \rightarrow \mathrm{b}_{\pi'})$. 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

$$\forall\pi.\, \exists\pi'.\, pc_{\pi'} \wedge (\neg pc_{\pi} \rightarrow \bigcirc\Box(s_{\pi} \rightarrow \bigcirc v_{\pi'})) \ . \tag{4.11}$$

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 4 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 4.14 *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*

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

Proof The $\Rightarrow$ direction is the same as in proofs of Theorem 4.11 and Lemma 4.10.

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

- Case $T' \in \mathcal{L}(\varphi)$, then for all $t_1 \in T'$ there is a $t_2 \in T'$ such that $\Pi_{fin} = \{\pi \mapsto t_1, \pi' \mapsto t_2\} \vDash_\emptyset \psi$. Especially, for t, there is a corresponding trace t^* such that $\{\pi \mapsto t, \pi' \mapsto t^*\} \vDash_\emptyset \psi$, thus $t^* \in \mathcal{L}(\mathcal{M}_\varphi[t/\pi])$. From (a) it follows that $t^* \in \mathcal{L}(\mathcal{M}_\varphi[t'/\pi])$. Hence, $\{\pi \mapsto t', \pi' \mapsto t^*\} \vDash_\emptyset \psi$ and thereby $T' \cup \{t'\} \in \mathcal{L}(\varphi)$.
- Case $T' \cup \{t'\} \in \mathcal{L}(\varphi)$, then for all $t_1 \in T' \cup \{t'\}$ there is a $t_2 \in T' \cup \{t'\}$ such that $\{\pi \mapsto t_1, \pi' \mapsto t_2\} \vDash_\emptyset \psi$. Assume for the sake of contradiction there is a $t_1 \in T'$ such that there is no $t_2 \in T'$ with $\{\pi \mapsto t_1, \pi' \mapsto t_2\} \vDash_\emptyset \psi$. It follows that $\{\pi \mapsto t_1, \pi' \mapsto t'\} \vDash_\emptyset \psi$, i.e., $t_1 \in \mathcal{L}(\mathcal{M}_\varphi[t'/\pi'])$. From (b) it follows that $t_1 \in \mathcal{L}(\mathcal{M}_\varphi[t/\pi'])$, leading to the contradiction that $\{\pi \mapsto t_1, \pi' \mapsto t\} \vDash_\emptyset \psi$ and $t \in T'$. Hence, $T' \in \mathcal{L}(\varphi)$. ∎

Corollary 4.15 *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*

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

We show the effect of the dominance characterization on two example formulas. Consider the HyperLTL formula $\forall\pi.\,\exists\pi'.\,\square(a_\pi \rightarrow b_{\pi'})$ 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 $\exists\pi.\,\forall\pi'.\,\square(a_\pi \rightarrow b_{\pi'})$. In this case, $\{a\}\emptyset$ dominates $\{a\}\{a\}$ and $\{b\}\emptyset$ dominates $\{b\}\{b\}$.

For our conference management example formula given in Equation 4.11, 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.

4.6 Evaluation

In this section, we report on experimental results of the presented techniques. We evaluated a prototype implementation presented in [23]. 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 4.3 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.

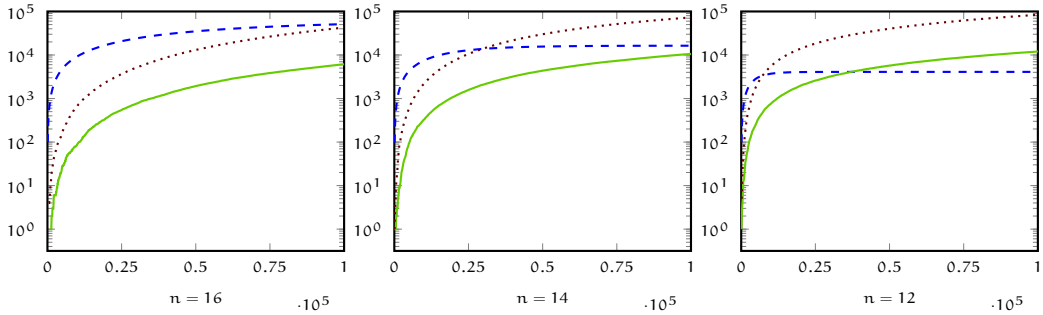

Figure 4.3: Absolute numbers of violations in red (dotted), number of instances stored in blue (dashed), number of instances pruned in green (solid) for 10^5 randomly generated traces of length 100000. The y axis is scaled logarithmically.

In Fig. 4.4 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 the trace analysis technique. 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 our technique results in a tremendous speed up of the monitoring algorithm.

4.7 An Optimization: Analyzing Specifications with EAHyper

As a preprocessing step to our monitoring algorithm, EAHyper was used in [23] to detect whether a formula is **symmetric**, **transitive**, or **reflexive**. This optimization and its possible application in verifying systems against hyperproperties will be out of the scope of this thesis and the interested reader may be referred to [23]. However, we briefly discuss the proposed idea of the specification analysis here as a proof of the practical relevance of a HyperLTL-SAT solver in modern verification techniques.

If a hyperproperty is symmetric, every symmetric monitor instantiation can be omitted, hence, only halve of the language membership tests have to be performed. Consider, again, the following formalization of observational determinism, which is symmetric:

$$\forall\pi.\forall\pi'.\,(O_\pi = O_{\pi'})\,W\,(I_\pi \neq I_{\pi'})$$

If a hyperproperty is transitive, we can check every incoming trace against a reference trace, i.e., only one monitor is needed. The canonical example of a transitive

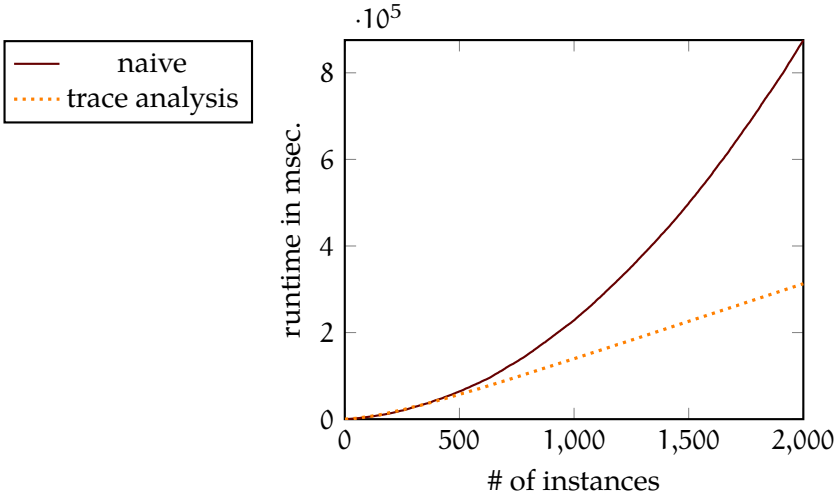

Figure 4.4: Runtime comparison of naive monitoring approach with a version using specification analysis, trace analysis, and a combination of both.

hyperproperty is equality:

$$\forall\pi.\forall\pi'.\ \square(a_\pi \leftrightarrow a_{\pi'})$$

Both hyperproperties above are reflexive. For such formulas the monitor where every trace variable is initialized with the same trace may be omitted. Whether a hyperproperty formalized in the alternation-free fragment of HyperLTL fulfils one of the properties above can be quickly checked with EAHyper, even for formulas of non trivial size [23]. This shows, that EAHyper can be effectively used in practice to speed up verification processes by analyzing and, thus, understanding formalized hyperproperties.

This concludes our section on the runtime verification problem of HyperLTL. We developed a practical, automata-based monitoring algorithm for which we presented a trace storage minimization algorithm.

Chapter 5

Conclusion

In this thesis, we presented an automata based monitoring approach for HyperLTL. We showed that deciding whether an alternation-free HyperLTL formula is monitorable is PSpace-complete. We provided two algorithms: an offline and an online algorithm for monitoring HyperLTL formulas which lie in the monitorable fragment. Furthermore, we presented an offline algorithm for monitoring HyperLTL formulas with one quantifier alternation.

We introduced EAHyper, the first satisfiability solver for hyperproperties expressed in the $\exists^*\forall^*$ fragment of HyperLTL. We sketched a specification analysis [23] based on EAHyper, which reduces the algorithmic workload by reducing the number of comparisons between a newly observed trace and the previously stored traces.

To handle the potentially unbounded memory consumption, we presented a trace analysis technique that reduces the traces that must be stored to a minimum. We briefly evaluated the minimization algorithm on a prototype implementation, showing that our technique can tremendously reduce the memory consumption during the monitoring process.

Future work includes the development of a practical tool for the monitoring of HyperLTL formulas based on our approach. Furthermore, developing an enforcement mechanism for (a class of) HyperLTL formulas will be a challenging and interesting task for the future.

Bibliography

[1] Shreya Agrawal and Borzoo Bonakdarpour. Runtime verification of k-safety hyperproperties in HyperLTL. In **Proceedings of CSF**, pages 239–252. IEEE Computer Society, 2016.

[2] Aslan Askarov and Andrei Sabelfeld. Tight enforcement of information-release policies for dynamic languages. In **Proceedings of CSF**, pages 43–59. IEEE Computer Society, 2009.

[3] Thomas H. Austin and Cormac Flanagan. Permissive dynamic information flow analysis. In **Proceedings of PLAS**, page 3. ACM, 2010.

[4] Andreas Bauer. Monitorability of omega-regular languages. **CoRR**, abs/1006.3638, 2010.

[5] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. **ACM Trans. Softw. Eng. Methodol.**, 20(4):14:1–14:64, 2011.

[6] Abhishek Bichhawat, Vineet Rajani, Deepak Garg, and Christian Hammer. Information flow control in webkit's javascript bytecode. In **Proceedings of POST**, volume 8414 of **LNCS**, pages 159–178. Springer, 2014.

[7] Borzoo Bonakdarpour and Bernd Finkbeiner. Runtime verification for HyperLTL. In **Proceedings of RV**, volume 10012 of **LNCS**, pages 41–45. Springer, 2016.

[8] Noel Brett, Umair Siddique, and Borzoo Bonakdarpour. Rewriting-based runtime verification for alternation-free HyperLTL. In **Proceedings of TACAS**, volume 10206 of **LNCS**, pages 77–93, 2017.

[9] Andrey Chudnov, George Kuan, and David A. Naumann. Information flow monitoring as abstract interpretation for relational logic. In **Proceedings of CSF**, pages 48–62. IEEE Computer Society, 2014.

[10] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantified interference

for a while language. **Electronic Notes in Theoretical Compututer Science**, 112:149–166, 2005.

[11] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. **Journal of Computer Security**, 18(6):1157–1210, 2010.

[12] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In **Proceedings of POST**, volume 8414 of **LNCS**, pages 265–284. Springer, 2014.

[13] Marcelo d'Amorim and Grigore Rosu. Efficient monitoring of omega-languages. In **Proceedings of CAV**, volume 3576 of **LNCS**, pages 364–378. Springer, 2005.

[14] Dominique Devriese and Frank Piessens. Noninterference through secure multi-execution. In **Proceedings of SP**, pages 109–124. IEEE Computer Society, 2010.

[15] Rayna Dimitrova, Bernd Finkbeiner, Máté Kovács, Markus N. Rabe, and Helmut Seidl. Model checking information flow in reactive systems. In **Verification, Model Checking, and Abstract Interpretation - 13th International Conference, VMCAI 2012, Philadelphia, PA, USA, January 22-24, 2012. Proceedings**, pages 169–185, 2012.

[16] Rayna Dimitrova, Bernd Finkbeiner, and Markus N. Rabe. Monitoring temporal information flow. In **Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change - 5th International Symposium, ISoLA 2012, Heraklion, Crete, Greece, October 15-18, 2012, Proceedings, Part I**, pages 342–357, 2012.

[17] Alexandre Duret-Lutz. Manipulating LTL formulas using spot 1.0. In **Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Proceedings**, pages 442–445, 2013.

[18] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In **Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003, Selected Revised Papers**, pages 502–518, 2003.

[19] Bernd Finkbeiner and Christopher Hahn. Deciding hyperproperties. In **Proceedings of CONCUR**, volume 59 of **LIPIcs**, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[20] Bernd Finkbeiner and Henny Sipma. Checking finite traces using alternating automata. **Formal Methods in System Design**, 24(2):101–127, 2004.

[21] Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In **Proceedings of CAV**, volume 9206 of **LNCS**, pages 30–48. Springer, 2015.

[22] Bernd Finkbeiner, Christopher Hahn, and Marvin Stenger. 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**, pages 564–570, 2017.

[23] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. In **Runtime Verification - 17th International Conference, RV 2017, Seattle, WA, USA, September 13-16, 2017. Proceedings, to appear**, 2017.

[24] Gurvan Le Guernic, Anindya Banerjee, Thomas P. Jensen, and David A. Schmidt. Automata-based confidentiality monitoring. In **Proceedings of ASIAN**, volume 4435 of **LNCS**, pages 75–89. Springer, 2006.

[25] Christopher Hahn. Deciding hyperltl. In **Bachelor's Thesis**, 2016.

[26] Richard W. Hamming. Error detecting and error correcting codes. **Bell Labs Technical Journal**, 29(2):147–160, 1950.

[27] Klaus Havelund and Grigore Rosu. Efficient monitoring of safety properties. **STTT**, 6(2):158–173, 2004.

[28] Máté Kovács and Helmut Seidl. Runtime enforcement of information flow security in tree manipulating processes. In **Proceedings of ESSoS**, volume 7159 of **LNCS**, pages 46–59. Springer, 2012.

[29] Martin Leucker and Christian Schallhart. A brief account of runtime verification. **J. Log. Algebr. Program.**, 78(5):293–303, 2009.

[30] Jianwen Li, Lijun Zhang, Geguang Pu, Moshe Y. Vardi, and Jifeng He. LTL satisfiability checking revisited. In **2013 20th International Symposium on Temporal Representation and Reasoning, TIME 2013**, pages 91–98, 2013.

[31] Zohar Manna and Amir Pnueli. **Temporal verification of reactive systems - safety**. Springer, 1995. ISBN 978-0-387-94459-3.

[32] John McLean. Proving noninterference and functional correctness using traces. **Journal of Computer Security**, 1(1):37–58, 1992.

[33] Stefan Mitsch and André Platzer. Modelplex: verified runtime validation of verified cyber-physical system models. **Formal Methods in System Design**, 49(1-2):33–74, 2016.

[34] Minh Ngo, Fabio Massacci, Dimiter Milushev, and Frank Piessens. Runtime enforcement of security policies on black box reactive programs. In **Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015**, pages 43–54, 2015.

[35] Amir Pnueli. The temporal logic of programs. In **18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977**, pages 46–57, 1977.

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

[37] Thomas Reinbacher, Kristin Yvonne Rozier, and Johann Schumann. Temporal-logic based runtime observer pairs for system health management of real-time systems. In **Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings**, pages 357–372, 2014.

[38] A. W. Roscoe. CSP and determinism in security modelling. In **Proceedings of the 1995 IEEE Symposium on Security and Privacy**, pages 114–127, 1995.

[39] Carl Martin Rosenberg, Martin Steffen, and Volker Stolz. Leveraging dtrace for runtime verification. In **Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings**, pages 318–332, 2016.

[40] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. **IEEE Journal on Selected Areas in Communications**, 21(1):5–19, 2003.

[41] Stefan Schwendimann. A new one-pass tableau calculus for PLTL. In **Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX '98 Proceedings**, pages 277–292, 1998.

[42] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. In **Proceedings of STOC**, pages 159–168. ACM, 1982.

[43] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. **J. ACM**, 32(3):733–749, 1985.

[44] Geoffrey Smith. On the foundations of quantitative information flow. In **Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Proceedings**, pages 288–302, 2009.

[45] G. Edward Suh, Jae W. Lee, David Zhang, and Srinivas Devadas. Secure program execution via dynamic information flow tracking. In **Proceedings of ASPLOS**, pages 85–96. ACM, 2004.

[46] Deian Tabakov, Kristin Y. Rozier, and Moshe Y. Vardi. Optimized temporal monitors for systemc. **Formal Methods in System Design**, 41(3):236–268, 2012.

[47] Mathy Vanhoef, Willem De Groef, Dominique Devriese, Frank Piessens, and Tamara Rezk. Stateful declassification policies for event-driven programs. In **Proceedings of CSF**, pages 293–307. IEEE Computer Society, 2014.

[48] Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. In **16th IEEE Computer Security Foundations Workshop CSFW-16 2003**, page 29, 2003.