# Deciding Hyperproperties Combined with Functional Specifications

Raven Beutner
CISPA Helmholtz Center for Information Security
Germany

David Carral
LIRMM, Inria, University of Montpellier, CNRS
France

Bernd Finkbeiner
CISPA Helmholtz Center for Information Security
Germany

Jana Hofmann
CISPA Helmholtz Center for Information Security
Germany

Markus Krötzsch
Technische Universität Dresden
Germany

## ABSTRACT

We study satisfiability for HyperLTL with a $\forall^*\exists^*$ quantifier prefix, known to be highly undecidable in general. HyperLTL can express system properties that relate multiple traces (so-called *hyperproperties*), which are often combined with *trace properties* that specify functional behavior on single traces. Following this conceptual split, we first define several *safety* and *liveness* fragments of $\forall^*\exists^*$ HyperLTL, and characterize the complexity of their (often much easier) satisfiability problem. We then add LTL trace properties as functional specifications. Though (highly) undecidable in many cases, this way of combining "simple" HyperLTL and arbitrary LTL also leads to interesting new decidable fragments. This systematic study of $\forall^*\exists^*$ fragments is complemented by a new (incomplete) algorithm for $\forall\exists^*$-HyperLTL satisfiability.

## CCS CONCEPTS

• **Theory of computation → Logic and verification**; **Modal and temporal logics**.

## KEYWORDS

Hyperproperties, HyperLTL, Satisfiability

**ACM Reference Format:**
Raven Beutner, David Carral, Bernd Finkbeiner, Jana Hofmann, and Markus Krötzsch. 2022. Deciding Hyperproperties Combined with Functional Specifications. In *37th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (LICS '22), August 2–5, 2022, Haifa, Israel.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3531130.3533369

## 1 INTRODUCTION

Hyperproperties are properties that relate multiple execution traces of a system [14] and comprise a range of relevant properties from many areas of computer science. Examples are symmetry, optimality, robustness, and noninterference. The most prominent logic for expressing hyperproperties is HyperLTL [13], which extends LTL with trace quantification. Generalized noninterference [34], for example, states that high-security inputs do not influence the input-output behavior observable by a low-security user, which can be expressed in HyperLTL as follows.

$$\forall\pi\forall\pi'\exists\pi''.\,\square\Big(\bigwedge_{a\in L_{out}\cup L_{in}}(a_\pi \leftrightarrow a_{\pi''}) \wedge \bigwedge_{a\in H_{in}}(a_{\pi'} \leftrightarrow a_{\pi''})\Big)$$

The formula states that for every two traces $\pi, \pi'$, there exists a trace $\pi''$ that combines the low-security inputs and outputs on $\pi$ and the high-security inputs on $\pi'$.

In this paper, we study the satisfiability problem of HyperLTL. For LTL, satisfiability is PSPACE-complete [40]. For hyperproperties, satisfaction cannot be decided by analyzing single traces in isolation, making formal reasoning challenging. Deciding satisfiability in the $\exists^*\forall^*$ fragment of HyperLTL is already EXPSPACE-complete [20]; and deciding hyperproperties with a $\forall^*\exists^*$ trace quantifier alternations is, in general, strongly undecidable, namely $\Sigma^1_1$-complete [25]. Nevertheless, the $\forall^*\exists^*$ fragment contains many relevant properties like generalized noninterference, program refinement, and software doping [16, 34]. Despite its importance, positive results for the $\forall^*\exists^*$ fragment have been very rare and were only obtained by heavy restrictions on the use of temporal operators or by assuming finite models [33] (see related work below). Algorithms, even if incomplete, are similarly missing.

In this work, we address these shortcomings by studying ways of solving satisfiability of $\forall^*\exists^*$ HyperLTL specifications. We identify simple yet expressive fragments of $\forall^*\exists^*$ with better computational properties, where our approach derives interesting fragments in two steps. First, we split a specification into hyperproperty and trace property, so that we can focus on "simple" hyperproperties. Second, to find such simple hyperproperties, we systematically study fragments of temporal safety and temporal liveness hyperproperties. This work towards new decidable fragments is complemented by a new (incomplete but often successful) algorithm that is applicable to arbitrary $\forall\exists^*$ specifications.

*Splitting in Hyperproperties and Trace Properties.* So far, all HyperLTL decidability results were obtained by considering HyperLTL specifications in isolation. Most of the time, however, specifications refer to a specific system. The hyperproperty itself is often relatively simple (like the noninterference property above) and only


*LICS '22, August 2–5, 2022, Haifa, Israel*

ACM ISBN 978-1-4503-9351-5/22/08.
https://doi.org/10.1145/3531130.3533369

gets difficult to satisfy given a specification of the functional behavior of the system.[1] The following example highlights this interplay between functional property and hyperproperty.

*Example 1.1.* Consider a system of agents that send and receive data. Each trace describes the behavior of a single agent. We want the system to satisfy the following hyperproperty.

$$\varphi := \forall\pi\exists\pi'.\, \Diamond(send_\pi \wedge rec_{\pi'})$$

The formula states that each agent eventually sends its information and that there exists some agent receiving it. The formula on its own is easily satisfiable already by a one-trace model. In addition to the hyperproperty we add the simple functional specification (trace property)

$$\psi := (\neg rec)\,\mathcal{U}(rec \wedge \bigcirc\Box\neg rec) \wedge \Box(rec \leftrightarrow \bigcirc send)$$

which expresses that each agent receives data exactly once and sends it forth in the next step. Every model that satisfies the *combination* of $\varphi$ and $\psi$ needs to be infinite. Automatically checking satisfiability is thus complex as we cannot iteratively search for models of bounded (finite) size. ◁

A satisfiability checker that distinguishes between a functional specification and hyperproperties could be used to sanity-check whether a hyperproperty is satisfiable in combination with the specification of the system at hand.

*Temporal Safety and Temporal Liveness.* The classification into safety and liveness has a long tradition in the study of trace properties, where especially safety often allows for easier algorithms. For our analysis, we define analogous fragments: a HyperLTL formula is *temporal safety* (resp. *temporal liveness*) if its LTL body describes a safety (resp. liveness) property. We study the relationship to the existing notations of *hypersafety* and *hyperliveness* defined by Clarkson and Schneider [14]. Guided by our insights into the complete fragments, we derive several more specific classes of temporal safety and liveness properties, for which satisfiability is easier to decide.

*Main Results.* Our results are summarized in Table 1, where each line represents a class of HyperLTL properties, and the columns distinguish whether or not additional (arbitrarily complex) LTL specifications are allowed. All hardness results for $\forall^*\exists^*$ fragments, except in the NEXP cases, already hold for $\forall\exists^*$. The restriction to temporal safety makes the satisfiability of HyperLTL drop from $\Sigma^1_1$ to coRE, which we show by an effective reduction to satisfiability of first-order logic. While still undecidable, this enables the use of common first-order techniques such as resolution, tableaux, and related methods [37]. Hardness already holds for simple formulas consisting only of a single $\Box$ with $\bigcirc$s in its scope. If we add (non-safety) functional specifications, hardness jumps back to $\Sigma^1_1$. In contrast to temporal safety properties, the class of temporal liveness HyperLTL formulas is of analytical complexity, even without additional LTL specifications. However, again in contrast to $\Box(\bigcirc^*)$, formulas from the $\forall\exists^*.\,\Diamond(\bigcirc^*)$ fragment are decidable, even when combined with an arbitrary LTL specification. This is the first HyperLTL decidability result for formulas that can enforce

[1] Of course, we can incorporate the LTL property in the HyperLTL formula: we conceptually divide the specification into a (complicated) trace property and a (simple) hyperproperty.

**Table 1: Deciding satisfiability of HyperLTL specifications. All results, expect for decidability (dec.), denote completeness. Our notation is found in Section 2.3, e.g., $\forall^*\exists^*.\,\Box(\bigcirc^*)$ is the class of $\forall^*\exists^*$ formulas whose LTL body uses a single $\Box$ operator with optional $\bigcirc$ operators in its scope.**

| | | no LTL spec. | with LTL spec. |
|---|---|---|---|
| **Temporal Safety** | complete fragment | coRE [Thm. 3.7] | $\Sigma^1_1$ [Thm. 3.11] |
| | $\forall^*\exists^*.\,\bigcirc^*$ | NEXP [Thm. 3.12] | NEXP [Thm. 3.12] |
| | $\forall^*\exists^*.\,\Box$ | NEXP [Lem. 3.13] | $\Sigma^1_1$ [Thm. 3.11] |
| | $\forall^*\exists^*.\,\Box(\bigcirc^*)$ | coRE [Lem. 3.10] | $\Sigma^1_1$ [Thm. 3.11] |
| **Temporal Liveness** | complete fragment | $\Sigma^1_1$ [Thm. 4.2] | $\Sigma^1_1$ [Thm. 4.2] |
| | $\forall\exists^*$. det-liveness | trivial [Prop. 4.15] | $\Sigma^1_1$ [Cor. 4.16] |
| | $\forall\exists^*.\,\Diamond(\bigcirc^*)$ | NP [Lem. 4.4] | dec. [Thm. 4.6] |
| | $\forall^*\exists^*.\,\Diamond\wedge\cdots\wedge\Diamond$ | NP [Lem. 4.4] | $\Sigma^1_1$ [Thm. 4.12] |

models with infinitely many traces. The class also contains the specification from Example 1.1. This decidability result is tight in the sense that already conjunctions of multiple eventualities are analytical again.

Finally, to complement our decidability results, we propose a general approximation algorithm to find the *largest* model for specifications consisting of a HyperLTL formula and an LTL formula. Our experimental evaluation shows that our algorithm performs significantly better than approaches that iteratively search for models of bounded size [21, 33] and can even show unsatisfiability for many formulas (which is impossible in bounded approaches).

*Structure.* The remainder of this paper is structured as follows. We give some basic preliminaries and introduce HyperLTL in Section 2. We study the fragment of temporal safety in Section 3. We begin this study with the full fragment, and then gradually decrease in expressiveness all the way to the fragment containing only $\bigcirc$ operators. We then move to temporal liveness in Section 4. Analogous to the safety case, we begin with the full fragment, gradually decreasing to the fragment of pure eventualities, for which we establish decidability. Finally, in Section 5, we describe our approximation for finding the largest models and report on experimental results in Section 6.

*Related Work.* In recent years, many logics to express hyperproperties have been developed. Most approaches extend existing logics with trace or path quantification, examples besides HyperLTL are HyperCTL$^*$ [14], HyperQPTL [36], HyperPDL-$\Delta$ [27], and HyperATL$^*$ [7]. Monadic first-order logics can be extended by adding a special equal-level predicate [23] or using different types of quantifiers [4]. Recently, hyperproperties have also been obtained via a team semantics for trace logics [30, 43]. Apart from plain temporal logics, there are also hyperlogics for hyperproperties that are asynchronous [5, 12, 28], quantitative [22], or probabilistic [1, 17].

HyperLTL remains the most used among the proposed hyperlogics. Its satisfiability problem is known to be challenging: if we define fragments based on quantifier prefixes (but with an arbitrary body), then $\exists^*\forall^*$ is the most general fragment for which

satisfiability is still decidable (and EXPSPACE-complete), whereas $\forall\exists^*$ already leads to undecidability [20]. In fact, the $\forall^*\exists^*$ fragment is already satisfiability-complete: any HyperLTL formula can be effectively translated into equisatisfiable $\forall^*\exists^*$ formula [33]. Analyzing the case of (unrestricted) HyperLTL in more detail, Fortin et al. show satisfiability to be $\Sigma^1_1$-complete, and therefore above all problems in the arithmetic hierarchy [25]. In a more fine-grained analysis, Mascle and Zimmermann show that the problem becomes decidable if one only considers models of a bounded size or if, for selected quantifier prefixes, temporal operators are not nested [33]. In particular, $\forall\exists^*$ properties using only $\Diamond$ and $\Box$ (without $\bigcirc$s) are decidable (and always have a finite model), as no "diagonal" comparison between trace positions is possible [33]. The satisfiability of the logics HyperQPTL and HyperCTL$^*$, which both subsume HyperLTL, has been studied as well [15].

## 2 PRELIMINARIES

We assume a fixed, finite set of atomic propositions $AP$ and write $\Sigma \coloneqq 2^{AP}$. Given a symbol $\pi$, we write $AP_\pi$ for the set $\{a_\pi \mid a \in AP\}$. A trace $t$ is an element in $\Sigma^\omega$. For $i \in \mathbb{N}$, $t(i)$ denotes the $i$th element in $t$ (starting with the 0th) and $t[i,\infty]$ is the suffix of a trace starting in point in time $i$. For a finite trace $u \in \Sigma^*$ and an infinite trace $t \in \Sigma^\omega$, $u$ is a prefix of $t$ (written $u \Subset t$) if for every $0 \le i < |u|$, $u(i) = t(i)$. A trace property $P$ is a set of traces, whereas a hyperproperty $H$ is a set of sets of traces [14].

### 2.1 Trace Properties and LTL

Linear temporal logic (LTL) defines trace properties by combining temporal operators with boolean connectives. Its syntax is defined by the following grammar.

$$\psi \coloneqq a \mid \neg\psi \mid \psi \wedge \psi \mid \bigcirc\psi \mid \psi \mathcal{U} \psi$$

where $a \in AP$. We also use the standard Boolean connectives $\wedge$, $\rightarrow$, $\leftrightarrow$ and constants $\top$, $\bot$, as well as the derived LTL operators *eventually* $\Diamond\psi \coloneqq \top\mathcal{U}\psi$, and *globally* $\Box\psi \coloneqq \neg\Diamond\neg\psi$. The semantics of LTL is defined as usual.

$$\begin{array}{lll}
t \models a & \text{iff} & a \in t(0) \\
t \models \neg\psi & \text{iff} & t \not\models \psi \\
t \models \psi_1 \wedge \psi_2 & \text{iff} & t \models \psi_1 \text{ and } t \models \psi_2 \\
t \models \bigcirc\psi & \text{iff} & t[1,\infty] \models \psi \\
t \models \psi_1 \mathcal{U} \psi_2 & \text{iff} & \exists i.\, t[i,\infty] \models \psi_2 \text{ and } \forall j < i.\, t[j,\infty] \models \psi_1
\end{array}$$

Safety and liveness properties are prominent classes of trace properties [2]. Safety properties are characterized by the fact that each violation is caused after a finite time. Liveness properties characterize that something good happens eventually.

*Definition 2.1.* A property $P$ is *safety* if it holds that for every trace $t \notin P$, there exists a $u \Subset t$ such that for every $t'$ with $u \Subset t'$, we have $t' \notin P$. A property $P$ is *liveness* if for every $u \in \Sigma^*$, there exists a $t \in \Sigma^\omega$ with $u \Subset t$ and $t \in P$. ◁

### 2.2 Hyperproperties and HyperLTL

HyperLTL [13] extends LTL with explicit quantification over traces, thereby lifting it from a logic expressing trace properties to one expressing hyperproperties [14]. Let $\mathcal{V}$ be a set of trace variables. We define HyperLTL formulas with the following grammar.

$$\varphi \coloneqq \exists\pi.\, \varphi \mid \forall\pi.\, \varphi \mid \phi$$
$$\phi \coloneqq a_\pi \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc\phi \mid \phi \mathcal{U} \phi$$

Here, $\pi \in \mathcal{V}$ and $a \in AP$. We consider only closed formulas, i.e., formulas where for each atom $a_\pi$ the trace variable $\pi$ is bound by some trace quantifier. The semantics of HyperLTL is given with respect to a set of traces $T$ and a trace assignment $\Pi$, which is a partial mapping $\Pi : \mathcal{V} \rightharpoonup \Sigma^\omega$. For $\pi \in \mathcal{V}$ and $t \in T$, we write $\Pi[\pi \mapsto t]$ for the trace assignment obtained by updating the value of $\pi$ to $t$. We write $\Pi[i,\infty]$ for the assignment $\Pi[i,\infty](\pi) \coloneqq \Pi(\pi)[i,\infty]$.

$$\begin{array}{lll}
\Pi \models_T a_\pi & \text{iff} & a \in \Pi(\pi)(0) \\
\Pi \models_T \neg\phi & \text{iff} & \Pi \not\models_T \phi \\
\Pi \models_T \phi_1 \wedge \phi_2 & \text{iff} & \Pi \models_T \phi_1 \text{ and } \Pi \models_T \phi_2 \\
\Pi \models_T \bigcirc\phi & \text{iff} & \Pi[1,\infty] \models_T \phi \\
\Pi \models_T \phi_1 \mathcal{U} \phi_2 & \text{iff} & \exists i.\, \Pi[i,\infty] \models_T \phi_2 \text{ and } \forall j < i.\, \Pi[j,\infty] \models_T \phi_1 \\
\Pi \models_T \exists\pi.\, \varphi & \text{iff} & \exists t \in T.\, \Pi[\pi \mapsto t] \models_T \varphi \\
\Pi \models_T \forall\pi.\, \varphi & \text{iff} & \forall t \in T.\, \Pi[\pi \mapsto t] \models_T \varphi
\end{array}$$

We say that $T$ is a model of $\varphi$ (written $T \models \varphi$) if $\emptyset \models_T \varphi$, where $\emptyset$ denotes the empty trace assignment.

*Remark 2.2.* HyperLTL is closed under conjunction (and, more generally, under any boolean combination) [36]. For two HyperLTL formulas $\varphi_1, \varphi_2$, we therefore write $\varphi_1 \wedge \varphi_2$ for *some* HyperLTL formula expressing the conjunction of $\varphi_1, \varphi_2$. For examples $\big(\forall\pi\exists\pi'.\, \Box(a_\pi \leftrightarrow a_{\pi'})\big) \wedge \big(\forall\pi.\, \Diamond b_\pi\big)$ can be expressed as the HyperLTL formula $\forall\pi\forall\pi'\exists\pi''.\, \Box(a_\pi \leftrightarrow a_{\pi''}) \wedge \Diamond b_{\pi'}$. ◁

Analogous to trace properties, we can characterize hyperproperties as hypersafety and hyperliveness [14]. We lift the prefix relation $\Subset$ to sets of traces: a set $U \subseteq \Sigma^*$ of finite traces is a prefix of a set $T \subseteq \Sigma^\omega$ (written $U \Subset T$) if, for every $u \in U$, there exists a $t \in T$ such that $u \Subset t$.

*Definition 2.3.* A hyperproperty $H$ is *hypersafety* if for every $T \subseteq \Sigma^\omega$ with $T \notin H$, there exists a finite set $U \subseteq \Sigma^*$ with $U \Subset T$ such that, for every $T' \subseteq \Sigma^\omega$ with $U \Subset T'$, we have $T' \notin H$. A property $H$ is *hyperliveness* if for every finite set $U \subseteq \Sigma^*$, there exists $T \subseteq \Sigma^\omega$ with $U \Subset T$ and $T \in H$. ◁

Intuitively, a violation of a hypersafety property can be explained by the finite interaction of finitely many traces. Conversely, a hyperproperty is hyperliveness, if such a set can always be extended to a set satisfying the property.

### 2.3 Specifications and Notation

We study the combination of $\forall^*\exists^*$ HyperLTL formulas and arbitrary LTL formulas, and call such pairs *specifications*.

*Definition 2.4.* A *specification* is a pair $(\psi, \varphi)$ where $\psi$ is an LTL formula and $\varphi$ a HyperLTL formula. We say that $(\psi, \varphi)$ is satisfiable iff there exists a non-empty set of traces $T \subseteq \Sigma^\omega$ such that $\forall t \in T.\, t \models \psi$ and $T \models \varphi$. ◁

In general, we write $(\psi, \varphi)$ for specifications with arbitrary LTL and HyperLTL formulas. We use the following notation for fragments of specifications. We write $(\top, \varphi)$ to indicate that no LTL formula is given or, equivalently, the trace specification is *true*. We represent the quantifier prefix of the HyperLTL property using regular expressions. For example, $\forall\exists^*$ is a prefix consisting of a single universal quantifier followed by any number of existential quantifiers. We write $Q^*$ for an arbitrary prefix. The body of a HyperLTL formula is structured based on the use of temporal operators. We allow propositional (temporal-operator-free) formulas as conjuncts if not stated otherwise Consider the following example. A $\forall^*\exists^*.\,\square(\bigcirc^*)$ formula is of the form $\forall\pi_1 \dots \forall\pi_n \exists\pi_{n+1} \dots \exists\pi_{n+m}.\, (\square\phi) \wedge \phi'$, where $\phi$ may contain (potentially nested) $\bigcirc$ operators and $\phi'$ does not contain any temporal operators. Analogously, a formula in $\forall^*\exists^*.\,\square$ describes formulas as the one above but $\phi$ may not contain $\bigcirc$s. A formula in $\forall^*\exists^*.\,\Diamond \wedge \Diamond$ uses a conjunction of two eventualities (also without $\bigcirc$s).

## 2.4 Complexity of Undecidable Problems

Many problems considered in this paper are highly undecidable. To enable precise quantification of "how undecidable", we briefly recall the arithmetic and analytical hierarchy. We only provide a brief overview and refer to [38] for details. The arithmetic hierarchy contains all problems (languages) that can be expressed in first-order arithmetic over the natural numbers. It contains the class of recursively enumerable (RE) and co-enumerable problems (coRE) in its first level. The class $\Sigma^1_1$ (sitting in the analytical hierarchy) contains all problems that can be expressed with existential second-order quantification (over sets of numbers) followed by a first-order arithmetic formula. Analogously, the class $\Pi^1_1$ contains all problems expressible using universal second-order quantification. Consequently, both $\Sigma^1_1$ and $\Pi^1_1$ (strictly) contain the entire arithmetic hierarchy.

## 2.5 Machines

As a basic model of computation to show hardness we use two-counter machines. A *nondeterministic 2-counter machine* (2CM) consists of a finite set of instructions $l_1, \dots l_n$, which modify two counters $c_1, c_2$. Each instruction $l_i$ is of one of the following forms, where $x \in \{1, 2\}$ and $1 \leq j, k \leq n$.

- $l_i : \left[c_x \coloneqq c_x + 1;\ \texttt{goto}\ \{l_j, l_k\}\right]$
- $l_i : \left[c_x \coloneqq c_x - 1;\ \texttt{goto}\ \{l_j, l_k\}\right]$
- $l_i : \left[\texttt{if}\ c_x = 0\ \texttt{then goto}\ l_j\ \texttt{else goto}\ l_k\right]$
- $l_i$ : halt

Here, goto $\{l_j, l_k\}$ indicates that the machine nondeterministically chooses between instructions $l_j$ and $l_k$. A configuration of a 2CM is a tuple $(l_i, v_1, v_2)$, where $l_i$ is the next instruction to be executed, and $v_1, v_2 \in \mathbb{N}$ denote the values of the counters. The initial configuration of a 2CM is $(l_1, 0, 0)$. The transition relation between configurations is defined as expected. Decrementing a counter that is already 0 leaves the counter unchanged. A 2CM halts if a configuration with a halt instruction is reached. Deciding if a machine has a halting computation is RE-complete and deciding if it has an infinite computation is coRE-complete [35]. An infinite computation is *recurring* if it visits instruction $l_1$ infinitely many times. Deciding if a machine has a recurring computation, is $\Sigma^1_1$-hard [3, 24].

# 3 TEMPORAL SAFETY

In this section, we study the satisfiability problem of temporal safety HyperLTL formulas. We begin by defining temporal safety and argue why, compared to hypersafety, it is the more suitable fragment in the context of satisfiability. Subsequently, we show that temporal safety specifications improve the general $\Sigma^1_1$-hardness of $\forall^*\exists^*$ hyperproperties [25] to coRE-complete. We obtain this result by a reduction to satisfiability of first-order logic. In the next step, we investigate the combination of temporally safe hyperproperties with arbitrary functional trace specifications given in LTL. The complexity jumps again to $\Sigma^1_1$-completeness, perhaps surprisingly already for very basic $\forall\exists^*$ formulas only using one $\square$ as temporal operator. We, therefore, analyze the remaining fragments for decidability results and establish that hyperproperties that only use $\bigcirc$s as temporal operators are NEXPTIME-complete, even when adding arbitrary LTL specifications. The same holds for hyper-invariants (using $\square$) without an LTL specification.

## 3.1 Hypersafety and Temporal Safety

The safety fragment of LTL is one of the most successful fragments of temporal logics as it is amendable to easier monitoring and verification than arbitrary $\omega$-regular properties [31]. The concept of a safety property (i.e., every violation is caused after a finite time) naturally extends to hyperproperties, giving the general class of hypersafety (cf. Definition 2.3) [14]. However, hypersafety is not well suited for a systematic study of the decidability of hyperproperties. Deciding if a property is hypersafety is already highly undecidable and deciding if a hypersafety property is satisfiable is directly reducible to LTL satisfiability.

Proposition 3.1. *Deciding if a HyperLTL formula $\varphi$ is hypersafety is $\Pi^1_1$-hard.*

Proof. As shown in [19, Thm. 23], for any HyperLTL formula $\varphi$ we can effectively construct a formula $\varphi'$ such that $\varphi$ is unsatisfied iff $\varphi'$ is hypersafety. As HyperLTL unsatisfiability is $\Pi^1_1$-hard [25], the hardness follows. □

Proposition 3.2. *Given a HyperLTL formula $\varphi$ that is hypersafety, satisfiability of $\varphi$ is decidable in* PSPACE.

Proof. As hypersafety properties are closed under subsets [14], $\varphi$ is satisfiable iff it is satisfiable by a single trace model. Therefore, we can collapse all quantifiers in $\varphi$ to universal ones, giving an equisatisfiable (but not equivalent) $\forall^*$ formula for which satisfiability is decidable in PSPACE [20]. □

Instead of focusing on hypersafety, we study the satisfiability problem for a broader fragment of formulas which we call temporally safe.

*Definition 3.3.* A HyperLTL formula $Q\pi_1 \dots Q\pi_n.\, \phi$ is *temporal safety* if $\phi$ (interpreted as an LTL formula over $AP_{\pi_1} \cup \dots \cup AP_{\pi_n}$) describes a safety property. ◁

Similar to the case of LTL [31], the safety restriction on the body of the HyperLTL formula allows for easier *verification* (see, e.g., [8, 9]). We argue that temporal safety is also an interesting fragment to study in the context of *satisfiability*. First, compared with hypersafety, it is decidable whether a formula is temporally safe, as

safety is recognizable for LTL [39]. Second, the next two propositions show that temporal safety defines an expressive fragment: it subsumes all $\forall^*\exists^*$ hypersafety properties.

Proposition 3.4. *For any $\forall^*$ hypersafety property, there exists an equivalent $\forall^*$ property that is temporally safe.*

Proof. Let $\varphi = \forall \pi_1 \dots \pi_n . \phi$ be the hypersafety property. For any function $f : \{1, \dots, n\} \to \{1, \dots, n\}$ (of which there are $n^n$ many) we define the formula $\phi_{[f]}$ as the formula obtained by replacing each trace variable $\pi_i$ for $1 \leq i \leq m$ with $\pi_{f(i)}$. Define $\varphi' \coloneqq \forall \pi_1 \dots \pi_n . \phi'$ where

$$\phi' \coloneqq \bigwedge_{f:\{1,\dots,n\}\to\{1,\dots,n\}} \phi_{[f]}$$

It is easy to see that $\varphi \equiv \varphi'$ (using the semantics of universal quantification). We claim that $\phi'$ expresses a safety property when interpreted as trace property over $AP_{\pi_1} \cup \dots \cup AP_{\pi_n}$. Take any trace $t$ over $AP_{\pi_1} \cup \dots \cup AP_{\pi_n}$ with $t \not\models \phi'$ (as in the definition of safety, cf. Definition 2.1). Let $T = \{t_1, \dots, t_n\}$ be the set obtained by splitting $t$ into $n$ traces, i.e., $t_i$ is a trace over $AP$ that copies the assignments to $AP_{\pi_i}$ on $t$. By construction of $T$ we get $T \not\models \varphi'$ and, as $\varphi \equiv \varphi'$ is hypersafety, we get a finite set of finite traces $U \lessdot T$ such that no extension of $U$ satisfies $\varphi$. We assume that $U = \{u_1, \dots, u_n\}$ where $u_i \lessdot t_i$ for each $i$. This assumption is w.l.o.g., as we can replace multiple prefixes of the same $t_i$ by the longest among those prefixes, and add an arbitrary prefix of each $t_i$ that previously had no prefix in $U$. We further assume, again w.l.o.g., that all $u_i$s have the same length, say $k$. Now define $u$ as the finite trace (of length $k$) over $AP_{\pi_1} \cup \dots \cup AP_{\pi_n}$, where the assignment to $AP_{\pi_i}$ is taken from $u_i$. As $u_i \lessdot t_i$ for each $i$, we get $u \lessdot t$. It remains to argue that $u$ is a bad prefix of $\phi'$. Let $t'$ be any trace with $u \lessdot t'$. We, again, split $t'$ into traces $t'_1, \dots, t'_n$. Now $T' \coloneqq \{t'_1, \dots, t'_n\}$ satisfies $U \lessdot T'$, so $T' \not\models \varphi$. By the semantics of universal quantification, there thus exists a $f$ such that $[\pi_1 \mapsto t'_{f(1)}, \dots, \pi_n \mapsto t'_{f(n)}] \not\models \phi$ and so $[\pi_1 \mapsto t'_1, \dots, \pi_n \mapsto t'_n] \not\models \phi_{[f]}$. This implies that $t' \not\models \phi_{[f]}$ in the LTL semantics so $t' \not\models \phi'$ as required. □

*Remark 3.5.* We do not claim that every $\forall^*$ hypersafety property is temporally safe. Instead, Proposition 3.4 only states that there exists an equivalent temporally safe property. For example, $\forall\pi\forall\pi' . \Diamond(a_\pi \wedge \neg a_{\pi'})$ is unsatisfiable and thus hypersafety but $\Diamond(a_\pi \wedge \neg a_{\pi'})$ is not a safety property. ◁

Proposition 3.6. *For any $\forall^*\exists^*$ hypersafety property, there exists an equivalent $\forall^*$ property that is temporally safe.*

Proof. Let $\varphi = \forall \pi_1 \dots \pi_n \exists \pi'_1 \dots \pi'_m . \phi$ be hypersafety. For a function $g : \{1, \dots, m\} \to \{1, \dots, n\}$ we define the formula $\phi_{[g]}$ as the formula obtained by replacing each trace variable $\pi'_i$ for $1 \leq i \leq m$ with $\pi_{g(i)}$. Now define:

$$\varphi' \coloneqq \forall \pi_1 \dots \pi_n . \bigvee_{g:\{1,\dots,m\}\to\{1,\dots,n\}} \phi_{[g]}$$

We claim that $\varphi \equiv \varphi'$. Showing that $\varphi'$ implies $\varphi$ is easy as the disjunction gives an explicit witness for the existential quantifiers. For the other direction, assume $T \models \varphi$ for some model $T$. Let $t_1, \dots, t_n \in T$ be arbitrary. As $\varphi$ is a hypersafety property and $\{t_1, \dots, t_n\} \subseteq T$, we get that $\{t_1, \dots, t_n\} \models \varphi$. In particular, if we bind each $\pi_i$ to $t_i$ (in $\varphi$), we get witness traces $t'_1, \dots, t'_m \in \{t_1, \dots, t_n\}$ for the existential quantifiers in $\varphi$. Now define $g$ by mapping each $1 \leq j \leq m$ to $i \in \{1, \dots, n\}$ with $t'_j = t_i$. The trace assignment $[\pi_1 \mapsto t_1, \dots, \pi_n \mapsto t_n]$ satisfies $\phi_{[g]}$. As we can find such a $g$ for every $t_1, \dots, t_n \in T$, we get that $T \models \varphi'$ as required. As $\varphi'$ is a $\forall^*$ formula, we can conclude using Proposition 3.4. □

While temporal safety subsumes $\forall^*\exists^*$ hypersafety, it is a strictly larger fragment as shown by the following formula.

$$\left(\exists \pi . a_\pi\right) \wedge \left(\forall \pi . \Box(a_\pi \to \bigcirc\Box\neg a_\pi)\right) \wedge \left(\forall \pi \exists \pi' . \Box(a_\pi \leftrightarrow \bigcirc a_{\pi'})\right)$$

Every model of this property must contain *infinitely* many traces. Hypersafety properties, on the other hand, are closed under subsets and are therefore always satisfiable by a single trace model (if satisfiable at all) [14].

## 3.2 Temporal Safety without Functional Specifications

Having established that temporal safety spans a broad spectrum of properties, we now establish that the general analytical hardness of HyperLTL [25] drops to coRE-completeness for temporal safety.

Theorem 3.7. *The satisfiability problem for temporally safe HyperLTL is coRE-complete.*

We show the upper bound of Theorem 3.7 by giving an effective translation from temporally safe HyperLTL to first-order logic using the fact that satisfiability of first-order logic is coRE-complete [26]. Our translation enables the application of first-order satisfiability solvers in the realm of hyperproperties.

In our construction, we represent the body of a temporally safe HyperLTL formula as a safety automaton.

*Definition 3.8.* A safety automaton over alphabet $\Sigma$ is a tuple $\mathcal{A} = (Q, q_0, \delta)$ where $Q$ is a finite set of states, $q_0 \in Q$ a unquiet initial state, and $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation. A trace $t \in \Sigma^\omega$ is accepted by $\mathcal{A}$ if there exists *some* infinite run $r \in Q^\omega$ such that $r(0) = q_0$ and for all $i$, $(r(i), t(i), r(i+1)) \in \delta$. For every safety trace property $\phi$, there exists a safety automaton that accepts exactly those traces that satisfy $\phi$ [31]. ◁

Proposition 3.9. *The satisfiability problem of temporally safe HyperLTL is in coRE.*

Proof. Let $\varphi = Q_1\pi_1 \dots Q_n\pi_n . \phi$ be a temporally safe HyperLTL formula. Let $\mathcal{A}_\phi = (Q_\phi, q_{0,\phi}, \delta_\phi)$ be a safety automaton over $\Sigma \coloneqq 2^{AP_{\pi_1} \cup \dots \cup AP_{\pi_n}}$ that accepts $\phi$ (when interpreted as an LTL formula over $AP_{\pi_1} \cup \dots \cup AP_{\pi_n}$). We define in the following an equisatisfiable first-order formula $\Theta$, which can be computed from $\varphi$. For readability, we use two-sorted first-order logic, which is equisatisfiable to standard first-order logic. We use two sorts: *Trace*, which contains trace variables, and *TimePoint*, which contains time variables. We use the constant $i_0 : \mathit{TimePoint}$ to indicate the initial time point. The predicate $\mathit{Succ}(\cdot, \cdot)$ over $\mathit{TimePoint} \times \mathit{TimePoint}$ encodes the successor relation on time. For each $a \in AP$, we use a predicate $P_a(\cdot, \cdot)$ over $\mathit{Trace} \times \mathit{TimePoint}$ to indicate that on trace $t$, $a$ holds at point in time $i$. For each state $q \in Q_\phi$ we use a predicate $\mathit{State}_q$ over $\mathit{Trace}^n \times \mathit{TimePoint}$. Informally, $\mathit{State}_q(x_1, \dots, x_n, i)$ indicates that a run of $\mathcal{A}$ on traces $x_1, \dots, x_n$ can be in state $q$ at timepoint $i$.

We first ensure that each point in time has a successor and that the set of traces is non-empty.

$$\phi_{\text{succ}} \coloneqq \forall i : \mathit{TimePoint}.\, \exists i' : \mathit{TimePoint}.\, \mathit{Succ}(i, i')$$
$$\phi_{\text{non-empty}} \coloneqq \exists x : \mathit{Trace}.\, \top$$

For each state $q \in Q_\phi$, we construct a formula $\rho_q$ (over free variables $x_1, \dots, x_n$), describing that, for any choice of traces and at any point in time, there is a transition in $\mathcal{A}$ to some successor state.

$$\rho_q \coloneqq \forall i, i' : \mathit{TimePoint}.\, \mathit{State}_q(x_1, \dots, x_n, i) \wedge \mathit{Succ}(i, i') \rightarrow \bigvee_{(q,\sigma,q') \in \delta_\phi} \Big( \bigwedge_{a_{\pi_j} \in \sigma} P_a(x_j, i) \wedge \bigwedge_{a_{\pi_j} \notin \sigma} \neg P_a(x_j, i) \wedge \mathit{State}_{q'}(x_1, \dots, x_n, i') \Big)$$

Now, define $\Theta$ as follows

$$\Theta \coloneqq Q_1 x_1 : \mathit{Trace}. \dots Q_n x_n : \mathit{Trace}.\, \phi_{\text{succ}} \wedge \phi_{\text{non-empty}} \wedge \bigwedge_{q \in Q} \rho_q \wedge \mathit{State}_{q_0}(x_1, \dots, x_n, i_o).$$

The last conjunct ensures that all trace tuples chosen by the quantifiers have an infinite run in $\mathcal{A}$ starting in the initial state and in the initial time point. If $\Theta$ is satisfiable, we can construct a trace assignment by setting the propositions based on the evaluation of $P_a(\cdot, \cdot)$ in a satisfying first-order model of $\Theta$ and vice versa. A detailed proof can be found in the full version [6]. Note that our construction is limited to safety automata. If we were to use, e.g. Büchi automata, we could not ensure infinitely many visits to an accepting state. □

To complement the upper bound, we show coRE-hardness by reducing the complement of the halting problem of deterministic Turing machines. The proof shows that already a *single* □ with nested ◯s suffices for coRE-hardness.

Lemma 3.10. *The satisfiability problem is* coRE*-hard for specifications* $(\top, \varphi)$ *where* $\varphi$ *is of the form* $\forall\exists^*.\, \square(\bigcirc^*)$.

Proof Sketch. We encode the non-termination of deterministic Turning machines, which is coRE-hard. Each trace represents a configuration of the machine and the $\forall\exists$ formula demands that each configuration encoded in trace $\pi$ has a successor configuration on some trace $\pi'$. As the transitions of a TM can be checked locally, we can encode a successor configuration by comparing every three consecutive positions on $\pi$ and $\pi'$, which is possible with a single globally. We give a detailed proof in the full version [6]. □

This completes the proof of Theorem 3.7.

## 3.3 Temporal Safety with Functional Specifications

We now investigate satisfiability for the combination of temporally safe HyperLTL formulas and LTL properties. If the LTL specification is safety, we can simply merge the trace property with the temporally safe hyperproperty, maintaining the applicability of Theorem 3.7. The situation changes if we allow non-safety trace properties.

Theorem 3.11. *The satisfiability problem is* $\Sigma^1_1$*-hard for specifications* $(\psi, \varphi)$ *where* $\varphi$ *is of the form* $\forall\exists^*.\, \square$.

Proof Sketch. To show $\Sigma^1_1$-hardness, we encode recurring computations of nondeterministic two-counter machines. We represent each configuration by encoding the current instruction and two atomic propositions $\mathbf{c}_1, \mathbf{c}_2$ that hold exactly once, i.e., counter $x$ has value $v$ if $\mathbf{c}_x$ occurs in the $v$th position. To ensure a recurring computation, we add a third counter $t$ that decreases in each step. When it reaches 0, the trace must encode the initial instruction, at which point $t$ is reset to any value. The key idea of the proof is that each trace in the model represents two consecutive configurations, which are encoded over disjoint copies of $AP$ (for $i \in \{1, 2\}$, $AP^i = \{a^i \mid a \in AP\}$). In LTL, we can express that the second configuration encoded in a trace is a successor of the first configuration in that trace. Furthermore, we express in LTL that the value of $t$ either decreases or the initial instruction is executed. In the HyperLTL property we ensure the existence of the initial configuration, and state that for each trace $\pi$, there exist a $\pi'$ such that the second configuration on $\pi$ equals the first on $\pi'$. We can express the latter as

$$\forall\pi\exists\pi'.\, \square\Big( \bigwedge_{a \in AP} \big(a^2_\pi \leftrightarrow a^1_{\pi'}\big) \Big).$$

The resulting specification is satisfiable if and only if the machine has a recurring computation. We give a detailed proof in the full version [6]. □

## 3.4 Propositional Hyperproperties and Invariants

As we have seen, with arbitrary LTL properties present, a single □ operator suffices to jump to $\Sigma^1_1$. This leaves hyperproperties expressed using only ◯s as the only sub-analytical fragment. We settle the precise complexity of the resulting problem to be NEXPTIME-complete.

Theorem 3.12. *The satisfiability problem is* NEXPTIME*-complete for specifications* $(\psi, \varphi)$ *where* $\varphi$ *is of the form* $Q^*.\, \bigcirc^*$. *Hardness holds already for* $\psi = \top$, *a* $\forall^*\exists^*$ *prefix, and propositional* $\varphi$.

Proof Sketch. To show containment, we nondeterministically guess a set of finite traces $M \subseteq \Sigma^k$, where $k$ is the number of ◯ operators in the formula. We then verify that each trace in $M$ can be extended to one satisfying $\psi$ and that $M$ is a model of the hyperproperty. For the lower bound, we reduce the acceptance of an exponential-time bounded nondeterministic Turing machine to a HyperLTL formula. Our encoding is a $\forall^*\exists^*$-formula, which does not contain any temporal operators (not even ◯) and no trace property. Each trace in our model encodes a piece of information $(s, p, \gamma, q)$, where $s, p \in \mathbb{N}$, $\gamma$ is a tape symbol of the TM, and $q$ either a state of the TM or $\bot$. The tuple $(s, p, t, q)$ encodes that in time-step $s$ and at position $p$, the tape content is $\gamma$, and either the head is at position $p$ and the machine is in state $q$, or the head is not at position $p$ (if $q = \bot$). As the TM is time (and thus space) bounded, $s$ and $p$ are bound by $2^n$ for some $n$. We show that we can express in HyperLTL that the information encoded in a given model defines a valid accepting run of the TM. The resulting formula is satisfiable iff TM has an accepting computation. As we can never refer to all exponentially many positions explicitly, we use $\forall^*\exists^*$ formulas to encode a counter that references all positions. We give a formal proof in the the full version [6]. □

We note that HyperLTL without temporal operators has a strong connection to quantified boolean formulas (QBF), the validity of which is a standard PSPACE-complete problem [41]. In contrast to QBF, where the quantifier structure spans the polynomial hierarchy, our proof shows that in HyperLTL, the $\forall^*\exists^*$ fragment suffices to show NEXPTIME-hardness (refuting a conjecture in [33] that temporal-operator-free HyperLTL is equivalent to QBF). The reason is that HyperLTL satisfiability asks for the *existence* of some model for which the formula holds (which is related to the more general second-order QBF problem [32]).

If we forgo the additional trace property, we can also show the following lemma. Hardness already holds if we disallow propositional formulas outside of the $\square$.

Lemma 3.13. *The satisfiability problem is* NEXPTIME*-complete for specifications* $(\top, \varphi)$ *where* $\varphi$ *is of the form* $\forall^*\exists^*.\,\square$.

Proof. A property $\varphi = \forall^n\exists^m.\,(\square\phi) \wedge \phi'$ (where $\phi, \phi'$ do not contain any temporal operators) is satisfiable iff $\forall^n\exists^m.\,\phi \wedge \phi'$ is satisfiable. The result then follows using Theorem 3.12. □

## 4 TEMPORAL LIVENESS

The natural counterparts of safety properties are liveness properties, which postulate that "something good happens eventually". Similar to the case of hypersafety, hyperliveness as a fragment is not well-suited when studying satisfiability: any hyperliveness property is, by definition, satisfiable. Analogously to our study of temporal safety, we instead study HyperLTL properties whose body is a liveness property.

*Definition 4.1.* A HyperLTL formula $Q\pi_1 \dots Q\pi_n.\,\phi$ is a *temporal liveness* property if $\phi$ (interpreted as an LTL formula over $AP_{\pi_1} \cup \dots \cup AP_{\pi_n}$) describes a liveness property. ◁

We examine the temporal liveness fragment following the structure of Section 3 and point out analogous results wherever possible. As in Section 3, we first examine the entire class of temporal liveness and then gradually restrict this class to obtain new decidability results.

### 4.1 Hyperliveness and Temporal Liveness

As opposed to the safety case (cf. Proposition 3.6), temporal liveness and hyperliveness are incomparable. In temporal liveness, we can easily express falsity via $\forall\pi\forall\pi'.\,\Diamond(a_\pi \wedge \neg a_{\pi'})$, which is not hyperliveness. Conversely, the property $\forall\pi\exists\pi'.\,\square(a_\pi \leftrightarrow a_{\pi'})$ is hyperliveness (as we can always add more witness traces) but not expressible in temporal liveness.

### 4.2 General Temporal Liveness

Analogous to Theorem 3.7, we consider the full fragment of temporal liveness. Different from the fragment of temporal safety, the class of temporal liveness is already $\Sigma^1_1$-hard.

Theorem 4.2. *The satisfiability problem for* $\forall\exists^*$ *temporal liveness HyperLTL is* $\Sigma^1_1$*-hard.*

To prove Theorem 4.2, we show a stronger result: we can effectively reduce every $\forall^*\exists^*$ HyperLTL property to an equisatisfiable temporal liveness property.

Theorem 4.3. *Let* $(\psi, \varphi)$ *be a specification where* $\varphi$ *is of the form* $\forall^n\exists^m.\,\phi$*, and* $\psi$ *and* $\phi$ *are arbitrary but satisfiable LTL formulas. Then there is an effectively computable specification* $(\psi', \varphi')$ *such that* $\psi'$ *is an LTL liveness property,* $\varphi'$ *is a* $\forall^n\exists^m$ *temporal liveness property, and* $(\psi, \varphi)$ *and* $(\psi', \varphi')$ *are equisatisfiable.*

Proof. The idea is to move the start position of the formula under a $\Diamond$ operator. We introduce a fresh atomic proposition † and ensure that all traces satisfy the liveness property $\Diamond(\dagger \wedge \bigcirc\square\neg\dagger)$. The *unique* position where $\dagger \wedge \bigcirc\square\neg\dagger$ holds (the last time that † is true) is then the "start position" to evaluate the formula. Let $\varphi = Q^*.\,\phi$ where $Q^* = \forall\pi_1 \dots \pi_n\exists\pi_{n+1} \dots \pi_{n+m}$ is the quantifier prefix of $\varphi$. Define

$$\varphi' \coloneqq Q^*.\,\Diamond\left(\bigwedge_{i=1}^{n+m} \dagger_{\pi_i} \wedge \left(\bigcirc\square \bigwedge_{i=1}^{n+m} \neg\dagger_{\pi_i}\right) \wedge \phi\right)$$

In similar fashion, we define $\psi' \coloneqq \Diamond(\dagger \wedge (\bigcirc\square\neg\dagger) \wedge \psi)$. It is easy to see that both the LTL body of $\varphi'$ and $\psi'$ are liveness properties. Here it is crucial that we assumed that $\psi$ and $\phi$ are satisfiable.

We now claim that $(\psi, \varphi)$ is satisfiable if and only if $(\psi'\varphi')$ is satisfiable. For the first direction, assume that $T$ is a model for $(\psi, \varphi)$. The model with † added to the first step of all traces satisfies $(\psi', \varphi')$. For the other direction, let $T$ be a model of $(\psi', \varphi')$. We assume w.l.o.g. that there is no subset $T' \subsetneq T$ such that $T'$ is also a model for $(\psi', \varphi')$. The property enforces that for any traces $t_1, \dots, t_{n+m}$, where $t_{n+1}, \dots, t_{n+m}$ are the witness traces for $t_1, \dots, t_m$, † holds for the last time at a common time point. As $T'$ is minimal, every trace serves as a witness for some other traces. Therefore the last position where † holds is the same for all traces in $T'$. Let $i$ be this position. Then $\{t[i, \infty] \mid t \in T'\}$ is a model of $(\psi, \varphi)$. □

By Theorem 3.11, satisfiability of $\forall\exists^*$ HyperLTL is $\Sigma^1_1$-hard (note that we transform any specification $(\psi, \forall^n\exists^m.\,\phi)$ with $n \geq 1$ into an equivalent specification $(\top, \forall^n\exists^m.\,\phi')$ by integrating the trace property into the body of the HyperLTL formula). Theorem 4.3 thus gives a proof of Theorem 4.2. More generally, *every* HyperLTL formula can be effectively reduced to an equisatisfiable $\forall^2\exists^*$ HyperLTL formula [33, Thm. 5], so Theorem 4.3 shows that deciding temporal liveness can be used to decide full HyperLTL.

### 4.3 Simple Liveness Properties

The general class of temporal liveness thus does not define an "easier" fragment of HyperLTL. As in the case of safety properties, we study the precise boundary at which the jump to $\Sigma^1_1$ occurs by restricting to simpler forms of temporal liveness. Analogously to the case of invariants (described with $\square$), we study eventualities ($\Diamond$). Theorem 3.12 already shows that satisfiability is NEXPTIME-hard for propositional formulas. We therefore distinguish whether or not propositional conjuncts may occur outside the eventualities.

Lemma 4.4. *The satisfiability problem is* NP*-complete for specifications* $(\top, \varphi)$ *where* $\varphi$ *is of the form* $\forall^*\exists^*.\,\Diamond(\bigcirc^*) \wedge \dots \wedge \Diamond(\bigcirc^*)$ *and no propositional conjunct occurs outside of the* $\Diamond$ *operators. Hardness already holds for a single eventuality.*

Proof Sketch. We collapse all universal quantifiers in $\varphi$ and thereby reduce satisfiability of $(\top, \varphi)$ to boolean satisfiability. We give a detailed proof in the full version [6]. □

If we allow properties where propositional formulas occur outside of the $\Diamond$ operators, the complexity jumps back to NEXPTIME:

Lemma 4.5. *The satisfiability problem is* NEXPTIME*-complete for specifications* $(\top, \varphi)$ *where* $\varphi$ *is of the form* $\forall^*\exists^*.\, \Diamond(\bigcirc^*) \wedge \cdots \wedge \Diamond(\bigcirc^*)$ *(and we allow propositional conjuncts to occur outside of the* $\Diamond$ *operators).*

Proof. Assume we are given a formula $\varphi = \forall^*\exists^*.\, (\Diamond \phi_1) \wedge \cdots \wedge (\Diamond \phi_n) \wedge \phi'$ where $\phi_1, \ldots, \phi_n, \phi'$ contain only $\bigcirc$s. We get that $\varphi$ is satisfiable iff both $\forall^*\exists^*.\, (\Diamond \phi_1) \wedge \cdots \wedge (\Diamond \phi_n)$ and $\forall^*\exists^*.\, \phi'$ are satisfiable. The former is decidable in NP (see Lemma 4.4) and the latter in NEXPTIME (see Theorem 3.12), so the NEXPTIME upper bound follows. Hardness follows immediately from Theorem 3.12 (by simply ignoring the $\Diamond$s). □

It is worth to contrast these results with the analogous findings for simple temporally safe formulas. Lemma 4.4 shows that when adding an $\Diamond$ operator around a propositional formula, the problem drops from NEXPTIME (Theorem 3.12) to NP. This is in contrast to adding $\Box$ operators, which remains NEXPTIME-complete (Lemma 3.13). Invariants with nested $\bigcirc$ and propositional conjuncts are undecidable (Lemma 3.10), whereas eventualities with nested $\bigcirc$ operators and propositional conjuncts remain decidable (Lemma 4.5).

## 4.4 Eventualities with Functional Specifications

Surprisingly, the sharp contrast between $\Box$ and $\Diamond$ continues if we add functional specifications as LTL trace properties. For $\Box$, the resulting problem directly jumps to full analytical hardness (cf. Theorem 3.11). For $\Diamond$, we now show that the problem remains decidable. Our result reads as follows.

Theorem 4.6. *The satisfiability problem is decidable for specifications* $(\psi, \varphi)$ *where* $\varphi$ *is of the form* $\forall\exists^*.\, \Diamond(\bigcirc^*)$.

This result is interesting for two reasons. First, it outlines the precise difference between $\Box$ and $\Diamond$. Second, it defines a new decidable class that contains many properties of interest. In particular, formulas of the fragment can enforce infinite models.[2] For example, the specification in Example 1.1 falls in this newly identified fragment.

The remainder of this subsection provides a proof for Theorem 4.6. We introduce necessary concepts along the way.

*4.4.1 Eliminating Nexts.* We first show how to eliminate the $\bigcirc$ operators in $\varphi$.

Lemma 4.7. *Let* $(\psi, \varphi)$ *be a specification where* $\varphi$ *is of the form* $\forall^n\exists^m.\, \Diamond(\bigcirc^*)$. *There exists an effectively computable specification* $(\psi', \varphi')$ *where* $\varphi'$ *is the of the form* $\forall^n\exists^m.\, \Diamond$ *such that* $(\psi, \varphi)$ *and* $(\psi', \varphi')$ *are equisatisfiable.*

Proof Sketch. Let $\varphi = \forall^n\exists^m.\, (\Diamond \phi) \wedge \phi'$. We eliminate $\bigcirc$ operators in $\phi$ by letting traces range over tuples. Instead of considering traces in $\Sigma^\omega$, we consider traces in $(\Sigma^k)^\omega$, where $k$ is the lookahead needed to evaluate $\phi$ (which is upper bounded by the number of $\bigcirc$s in $\phi$). For each trace $t \in \Sigma^\omega$, we define $t' \in (\Sigma^k)^\omega$ by $t'(i) := (t(i), t(i+1), \ldots, t(i+k))$. This reduces the evaluation of $\phi$ to a formula without $\bigcirc$s. We also modify the LTL formula (which is allowed to contain $\bigcirc$ operators) to assert that the tuples in each tuple trace are consistent, i.e., for each step $i$ if $t(i) = (l_1, \ldots, l_k)$ then $t(i+1) = (l_2, \ldots, l_k, l_{k+1})$. A detailed proof can be found in the full version [6]. □

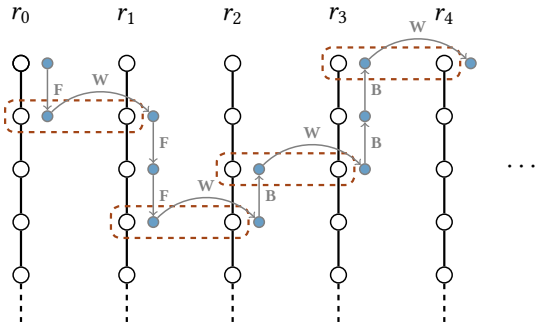


**Figure 1: Model for $\forall\pi\exists\pi'.\, \Diamond \phi$ formulas. Dashed boxes indicate the witness points for the $\Diamond$ operator.**

Using Lemma 4.7, we can assume that in Theorem 4.6, the HyperLTL formula $\varphi$ contains a single $\Diamond$ as the only temporal operator. For now, we make two further assumptions: First, we assume that $\varphi$ contains only a single $\exists$ quantifier, and, second, we assume that there are no additional propositional conjuncts outside the $\Diamond$. So let $\varphi = \forall\pi\exists\pi'.\, \Diamond \phi$ where $\phi$ contains no temporal operators. We begin by translating the trace property $\psi$ into a Büchi automaton.

*Definition 4.8.* A *state-labeled Büchi automaton* over alphabet $\Sigma$ is a tuple $\mathcal{A} = (Q, Q_0, \delta, F, L)$, where $Q$ is a finite set of states, $Q_0 \subseteq Q$ the initial states, $\delta \subseteq Q \times Q$ the transition relation, $F \subseteq Q$ the set of accepting states, and $L : Q \to \Sigma$ a state labeling function. An accepting run $r$ of $\mathcal{A}$ is an infinite sequence $r \in Q^\omega$ such that 1) $r(0) \in Q_0$, 2) $(r(i), r(i+1)) \in \delta$ for every $i$, and 3) $r(i) \in F$ for infinitely many $i$. The trace $L(r) \in \Sigma^\omega$ associated to a run is defined by applying $L$ pointwise, i.e., $L(r)(i) := L(r(i))$. For a set $X \subseteq Q$, we define $\mathit{Step}_{\mathcal{A}}(X) := \{q \in Q \mid \exists q' \in X.\, (q', q) \in \delta\}$ as all states reachable in one step from $X$ and $\mathit{Reach}_{\mathcal{A}}(n)$ as all states reachable in $n$ steps from a state in $Q_0$. ◁

Note that we use state-labeled automata (as opposed to transition-labeled automata) to simplify our construction. Let $\mathcal{A}_\psi = (Q_\psi, Q_{0,\psi}, \delta_\psi, F_\psi, L_\psi)$ be a (state-labeled) Büchi automaton over $2^{AP}$ accepting $\psi$ [42]. We say a state $q \in Q_\psi$ is *non-empty* if there exists an accepting infinite run starting in $q$. We assume that $\mathcal{A}_\psi$ only includes non-empty states. This is w.l.o.g. as we can remove all empty states without changing the language of $\mathcal{A}_\psi$. Detecting if a state is non-empty can be done easily using, e.g., nested depth-first search.

*4.4.2 Models for $\forall\exists$.* Intuitively, our decidability result can be derived as follows. Assume we had a model $T$ of $(\psi, \varphi)$. Let $R \subseteq Q_\psi^\omega$ be a set of accepting runs of $\mathcal{A}_\psi$ associated to $T$, i.e., $T = \{L_\psi(r) \mid r \in R\}$. As we consider a $\forall\exists$ formula, we can arrange the runs in $R$ as a sequence: we choose $r_0, r_1, \ldots \in R$ (not necessarily distinct)

[2] Existing decidability results for HyperLTL consider fragments that, if satisfiable, are satisfiable by a finite set of traces of bounded size. This includes the $\exists^*\forall^*$ fragment studied in [20] and the decidable fragments identified in [33].

such that, for each $i$, $r_{i+1}$ serves as a witness for $r_i$, i.e., $[\pi \mapsto L_\psi(r_i), \pi' \mapsto L_\psi(r_{i+1})] \models \Diamond\phi$. For two elements $\sigma, \sigma' \in \Sigma = 2^{AP}$, we write $\sigma \otimes \sigma' \models \phi$ if the assignment $\{a_\pi \mid a \in \sigma\} \cup \{a_{\pi'} \mid a \in \sigma'\} \in 2^{AP_\pi \cup AP_{\pi'}}$ satisfies $\phi$ (recall that $\phi$ contains no temporal operators so this is a simple propositional satisfaction check). We say $n_0, n_1, \dots$ are *witness points* for the sequence of runs $r_0, r_1, \dots$ if $L_\psi(r_i(n_i)) \otimes L_\psi(r_{i+1}(n_i)) \models \phi$ for every $i$, i.e., the $n_i$ point to a step at which the eventuality holds. The trace arrangement is depicted in Figure 1 (ignoring the blue smaller nodes and gray edges for now). For each $i$, the dashed box denotes the witness point $n_i$ where $r_i$ and $r_{i+1}$ satisfy $\phi$.

As an intermediate step, we describe an infinite-state Büchi system (a Büchi automaton without labels) that guesses such a "linear" model of $(\psi, \varphi)$. The states of the system are triples $(q, b, n)$, where $q \in Q_\psi$ is a state in $\mathcal{A}_\psi$, $b \in \{\Rrightarrow, \Lleftarrow\}$ gives a *running direction*, and $n \in \mathbb{N}$. Each state $(q, b, n)$ additionally satisfies $q \in Reach_{\mathcal{A}_\psi}(n)$. The initial states of the system are all states $(q_0, \Rrightarrow, 0)$ with $q_0 \in Q_{0,\psi}$. In each step, the system has three options: it can either run forwards, run backwards, or claim to have found a witness. In a forward step (**F**-step), the automaton moves from $(q, \Rrightarrow, n)$ to $(q', \Rrightarrow, n+1)$, where $(q, q') \in \delta_\psi$. Similarly, in a backwards step (**B**-step), it runs from $(q, \Lleftarrow, n+1)$ to $(q', \Lleftarrow, n)$, where $(q', q) \in \delta_\psi$. Note that in the backwards step, we always ensure that $q' \in Reach_{\mathcal{A}_\psi}(n)$. Lastly, the system can claim to have found a witness (**W**-step): if in state $(q, b, n)$, it can select any $q' \in Reach_{\mathcal{A}_\psi}(n)$ such that $L_\psi(q) \otimes L_\psi(q') \models \phi$. Afterward, the system continues in state $(q', b', n)$, where $b' \in \{\Rrightarrow, \Lleftarrow\}$ is chosen nondeterministically. Call the resulting system $\mathcal{S}$. We claim the following.

Lemma 4.9. *$\mathcal{S}$ has an infinite run that uses **W**-steps infinitely often if and only if $(\psi, \varphi)$ is satisfiable.*

Proof. We sketch both directions. For the "if" direction, assume there is a model for $(\psi, \varphi)$. We can arrange a subset of this model as depicted in Figure 1. Let $r_0, r_1, \dots$, with $r_i \in Q_\psi^\omega$ be the sequence of accepting runs in $\mathcal{A}_\psi$ and $n_i$ the witness points. Traversing the graph as shown by the small blue states in Figure 1 creates a run of $\mathcal{S}$. We start in $(r_0(0), \Rrightarrow, 0)$ and move forward (using **F**-steps) until the counter reaches $n_0$. At this point, we take the **W**-step from $(r_0(n_0), \Rrightarrow, n_0)$ to $(r_1(n_0), b, n_0)$, which is a valid edge in $\mathcal{S}$ by definition of the witness points. Afterward, we run towards counter value $n_1$, i.e., if $n_1 < n_0$, we set the running direction $b$ to $\Lleftarrow$ and otherwise to $\Rrightarrow$. We continue this procedure to construct an infinite run. For the example situation depicted in Figure 1, the resulting run would start with:

$$(r_0(0), \Rrightarrow, 0) \xrightarrow{\mathbf{F}} (r_0(1), \Rrightarrow, 1) \xrightarrow{\mathbf{W}} (r_1(1), \Rrightarrow, 1) \xrightarrow{\mathbf{F}} (r_1(2), \Rrightarrow, 2)$$
$$\xrightarrow{\mathbf{F}} (r_1(3), \Rrightarrow, 3) \xrightarrow{\mathbf{W}} (r_2(3), \Lleftarrow, 3) \xrightarrow{\mathbf{B}} (r_2(2), \Lleftarrow, 2) \xrightarrow{\mathbf{W}} \cdots$$

The resulting sequence is a run of $\mathcal{S}$ and uses **W**-steps infinitely many times.

For the "only if" direction, assume an infinite run $r = (q_0, b_0, m_0) \to (q_1, b_1, m_1) \to \cdots$ of $\mathcal{S}$. We split $r$ into infinitely many finite segments $x_0, x_1, \dots \in (Q_\psi \times \mathbb{N})^*$ by splitting each time $r$ takes a **W**-step (and discard the running direction). In the example run above we would get $x_0 = (r_0(0), 0)(r_0(1), 1)$, $x_1 = (r_1(1), 1)(r_1(2), 2)(r_1(3), 3), \dots$. In general, let $x_i$ be the sequence $(q_i^0, m_i^0) \dots (q_i^{k_i}, m_i^{k_i})$. From $x_i$, we construct a finite run $r_i \in Q_\psi^*$ of $\mathcal{A}_\psi$ starting in a state in $Q_{0,\psi}$ such that for every $0 \le j \le k_i$, $r_i(m_i^j) = q_i^j$. Using the fact that for each state $(q, b, n)$ in $\mathcal{S}$, we have $q \in Reach_{\mathcal{A}_\pi}(n)$, this is always possible. It is crucial that we cannot reverse the running direction between two **W**-steps. The finite $r_i$ ends in a state in $\mathcal{A}_\psi$, so by the assumption that all states are non-empty, we can extend it into an infinite accepting run $r_i' \in Q_\psi^\omega$. The set $\{L_\psi(r_0'), L_\psi(r_1'), \dots\}$ is a model of $(\psi, \varphi)$. □

*4.4.3 From Infinite State to Pushdown.* The construction of $\mathcal{S}$ requires infinitely many states as we need to carry the natural number $n$ to ensure valid **B** and **W** steps (which need access to $Reach_{\mathcal{A}_\psi}(n)$). We show next that we can replace this infinite state space by a finite pushdown system.

*Definition 4.10.* A *Büchi pushdown system* is a tuple $\mathcal{P} = (Q, \Gamma, Q_0, \gamma_0, \delta, F)$, where $Q$ is a finite set of states, $\Gamma$ a finite stack alphabet, $Q_0 \subseteq Q$ initial states, $\gamma_0 \in \Gamma$ the initial stack symbol, $\delta \subseteq (Q \times \Gamma^+) \times (Q \times \Gamma^*)$ a *finite* transition relation, and $F \subseteq Q$ a set of accepting states. The system operates on configuration $(q, \alpha)$, where $q \in Q$ and $\alpha \in \Gamma^*$. A transition $\langle q, \alpha\rangle \rightsquigarrow \langle q', \alpha'\rangle \in \delta$ describes that the system, if in state $q$ and $\alpha \in \Gamma^+$ is a prefix of the current stack, pops $\alpha$, pushes $\alpha' \in \Gamma^*$ to the stack and moves to state $q'$. An accepting run is an infinite sequence of configurations that starts in $(q_0, [\gamma_0])$ for some $q_0 \in Q_0$, respects $\delta$, and visits states in $F$ infinitely many times. It is decidable in polynomial time if a Büchi pushdown system has an accepting run [11]. ◁

We replace $\mathcal{S}$ with a pushdown system $\mathcal{P}$. Conceptually, we represent a state $(q, b, n)$ in $\mathcal{S}$ by the pushdown configuration with state $(q, b)$ and stack content $[Reach_{\mathcal{A}_\psi}(n), \dots, Reach_{\mathcal{A}_\psi}(0)]$, i.e., the length of the stack is $n+1$ and the $i$th element are all states reachable in $i$ steps. The states in the pushdown system thus have the form $(q, b)$ with $q \in Q_\psi, b \in \{\Rrightarrow, \Lleftarrow\}$ and the stack alphabet is $2^{Q_\psi}$. The initial stack symbol is $\gamma_0 := Q_{0,\psi}$ and the initial states are $\{(q_0, \Rrightarrow) \mid q_0 \in Q_{0,\psi}\}$. The transitions are of the following form:

$$(\mathbf{F})\ \frac{(q, q') \in \delta_\psi}{\big\langle (q, \Rrightarrow), [A]\big\rangle \rightsquigarrow \big\langle (q', \Rrightarrow), [Step_{\mathcal{A}_\psi}(A), A]\big\rangle}$$

$$(\mathbf{B})\ \frac{q' \in A_2 \qquad (q', q) \in \delta_\psi}{\big\langle (q, \Lleftarrow), [A_1, A_2]\big\rangle \rightsquigarrow \big\langle (q', \Lleftarrow), [A_2]\big\rangle}$$

$$(\mathbf{W})\ \frac{q' \in A \qquad L_\psi(q) \otimes L_\psi(q') \models \phi \qquad b, b' \in \{\Rrightarrow, \Lleftarrow\}}{\big\langle (q, b), [A]\big\rangle \rightsquigarrow \big\langle (q', b'), [A]\big\rangle}$$

Note the close correspondence with the transitions in $\mathcal{S}$. In particular, in **F**-steps, we compute $Reach_{\mathcal{A}_\psi}(n+1)$ based on $Reach_{\mathcal{A}_\psi}(n)$. In **B**-steps, the stack provides access to all states that are reachable, and thus guarantees the invariant that $q \in Reach_{\mathcal{A}_\psi}(n)$ for each state $(q, b, n)$ in $\mathcal{S}$. It is not hard to see that $\mathcal{P}$ has a run that uses **W**-steps infinitely often iff $\mathcal{S}$ has a run that uses **W**-steps infinitely often. Combined with Lemma 4.9 we thus get:

Lemma 4.11. *$\mathcal{P}$ has an accepting run that uses **W**-steps infinitely often if and only if $(\psi, \varphi)$ is satisfiable.*

Lastly, we can easily translate a Büchi pushdown system with transition-based acceptance (as in $\mathcal{P}$) to state-based acceptance (as

in Definition 4.10). Using the decidability of pushdown systems [11], we thus get that the satisfiability of $(\psi, \varphi)$ is decidable. Note that our proof gives an elementary upper bound of 2-EXPTIME (for $\forall\exists$ properties).[3]

*4.4.4 Propositional Conjuncts and $\forall\exists^*$.* We can now lift the two assumptions we made earlier. As a first extension, we modify our construction to also support formulas of the form $\forall\pi\exists\pi'.(\Diamond\phi)\wedge\phi'$. To do so, we keep track of the *first* state of the run we are currently considering. In a **W**-step, we then only select a witness state $q'$ that stems from an initial state which satisfies the propositional requirement $\phi'$ when combined with the initial state of the current run. We can access the set of all such states by keeping track of the set of states reachable from every individual state (by changing the stack alphabet to functions $Q_\psi \to 2^{Q_\psi}$).

As a second extension, we can show decidability for a $\forall\exists^m$ prefix by moving to *alternating* Büchi pushdown systems (defined as expected, see [11] for details). For $\forall\exists^m$, we can no longer arrange the traces of a model in a linear sequence (as depicted in Figure 1) and instead use $m$-ary trees labeled by traces such that the children of a node correspond to witness traces of that trace. In a **W**-step from a state $(q, b)$, we now select $m$ states $q_1, \dots, q_m$ (whereas we previously picked only $q'$) such that $q, q_1, \dots, q_m$ together satisfy $\phi$. Afterward, we need to find a new witnesses for *each* of the $q_i$. We accomplish this by introducing a *universal* transition that branches into states $(q_i, b_i)$ for each $1 \le i \le m$ (leaving the stack unchanged as before). The **F** and **B** steps stay purely nondeterministic. The resulting alternating pushdown system has an accepting run (which now has the form of a tree) iff $(\psi, \varphi)$ is satisfiable. As emptiness of alternating pushdown systems is still decidable (albeit only in exponential time) [11], we get a proof of Theorem 4.6 for the full $\forall\exists^*$-fragment. Note that, for the $\forall\exists^*$-fragment, our proof gives an elementary upper bound of 3-EXPTIME.

## 4.5 Conjunctions of Eventualities

We show that Theorem 4.6 is tight in the sense that already a conjunction of eventualities combined with an arbitrary trace property is undecidable (and even $\Sigma^1_1$-hard).

Theorem 4.12. *The satisfiability problem is $\Sigma^1_1$-hard for specifications $(\psi, \varphi)$ where $\varphi$ is of the form $\forall\exists^*.\Diamond\wedge\Diamond\wedge\Diamond$.*

Proof. We encode the problem of whether a nondeterministic 2CM with instructions $l_1, \dots, l_n$ has a recurring computation [3, 24]. Let $AP = \bigcup_{x\in\{1,2,t\}}\{\boxdot_x, \blacksquare_x, isZero_x\} \cup \{\mathbf{l}_1, \dots, \mathbf{l}_n\}$. Each trace encodes a configuration of the machine as follows. The current value of counter $x \in \{1, 2\}$ is encoded as a trace in $\emptyset^*\{\boxdot_x\}\{\blacksquare_x\}\emptyset^\omega$ such that the (unique) step at which $\boxdot_x$ holds indicates the current value of $c_x$. We later use proposition $\blacksquare_x$ (which always holds the step after $\boxdot_x$) to encode the update of the counter. The proposition $isZero_x$ holds exactly if the counter is zero. The current instruction is encoded by propositions $\{\mathbf{l}_1, \dots, \mathbf{l}_n\}$, of which exactly one holds globally along a trace. Finally, to ensure a recurring computation, we use a third counter $t$, which is encoded analogously to the counters above and counts down the steps to the next visit to $l_1$. It is easy to see that we can encode the validity of a configuration in an LTL formula $\psi$. For $x \in \{1, 2, t\}$ we ensure a valid counter via

$$(\neg\boxdot_x \wedge\neg\blacksquare_x)\mathcal{U}\big((\boxdot_x \wedge \neg\blacksquare_x) \wedge \bigcirc(\blacksquare_x \wedge \neg\boxdot_x) \wedge \bigcirc\bigcirc\Box(\neg\boxdot_x \wedge\neg\blacksquare_x)\big)$$

and correct placement of $isZero_x$ by $(\Box\, isZero_x) \vee (\Box\neg isZero_x)$ together with $isZero_x \leftrightarrow \boxdot_x$. Finally, we assert that the propositions $\{\mathbf{l}_1, \dots, \mathbf{l}_n\}$ are set correctly, i.e., $\bigvee_i \big(\Box\mathbf{l}_i \wedge \bigwedge_{j\neq i}\Box\neg\mathbf{l}_j\big)$. In the hyperproperty, we encode that there exists a trace representing the initial configuration as follows (note that we allow counter $t$ to have any value):

$$\varphi_{init} \coloneqq \exists\pi.(\mathbf{l}_1)_\pi \wedge (isZero_1)_\pi \wedge (isZero_2)_\pi$$

Lastly, we express that each trace has a successor. For each instruction $l_i$, we write $c(l_i) \in \{1, 2\}$ for the counter that is changed or tested in instruction $l_i$. We define $\overline{1} \coloneqq 2$ and $\overline{2} \coloneqq 1$ for the other counter. We then define

$$\begin{aligned}\varphi \coloneqq \forall\pi\exists\pi'.\Big(\Diamond \bigvee_{i\in\{1,\dots,n\}} (\mathbf{l}_i)_\pi \wedge exec(l_i)\Big) \wedge \\ \Big(\Diamond \bigvee_{i\in\{1,\dots,n\}} (\mathbf{l}_i)_\pi \wedge (\boxdot_{\overline{c(l_i)}})_\pi \wedge (\boxdot_{\overline{c(l_i)}})_{\pi'}\Big) \wedge \\ \Big(\Diamond\big((isZero_t)_\pi \wedge (\mathbf{l}_1)_\pi\big) \vee \big((\boxdot_t)_\pi \wedge (\blacksquare_t)_{\pi'}\big)\Big).\end{aligned}$$

Here, $exec(l_i)$ denotes that the action or test of instruction $l_i$ is performed on $c(l_i)$. For example, if $l_i : \big[c_x \coloneqq c_x{+}1;\ \texttt{goto}\ \{l_j, l_k\}\big]$, we define $exec(l_i)$ as

$$\big((\mathbf{l}_j)_{\pi'} \vee (\mathbf{l}_k)_{\pi'}\big) \wedge (\blacksquare_x)_\pi \wedge (\boxdot_x)_{\pi'}.$$

Note that $(\blacksquare_x)_\pi \wedge (\boxdot_x)_{\pi'}$ encodes that the counter $x$ is incremented. For a decrement operation, we can replace this with $(\blacksquare_x)_{\pi'} \wedge (\boxdot_x)_\pi$. If $l_i : \big[\texttt{if}\ c_x = 0\ \texttt{then goto}\ l_j\ \texttt{else goto}\ l_k\big]$, we define $exec(l_i)$ as

$$\begin{aligned}&(\boxdot_x)_\pi \wedge (\boxdot_x)_{\pi'} \wedge \\ &\big((isZero_x)_\pi \to (\mathbf{l}_j)_{\pi'}\big) \wedge \big(\neg(isZero_x)_\pi \to (\mathbf{l}_k)_{\pi'}\big).\end{aligned}$$

In $\varphi$, the first conjunct thus encodes that the counter $c(l_i)$ is updated and/or tested as required by $l_i$. The second conjunct states that the counter that is not involved in $l_i$ is left unchanged. As the current instruction is set consistently along a trace, both eventualities refer to the same instruction. Finally, the third conjunct ensures that the counter $t$ either decreases or is already zero, at which point the current instruction is $l_1$. In case the $t$-counter is zero, it can be reset to any value on $\pi'$. This ensures a recurring computation of the machine. It is easy to see that $(\psi, \varphi_{init} \wedge \varphi)$ is satisfiable iff the 2CM has a recurring computation (note that $\varphi_{init} \wedge \varphi$ is a $\forall\exists^2$-formula). □

While Theorem 4.12 requires three conjunctions of eventualities to show $\Sigma^1_1$-hardness, already two eventualities suffice to show undecidability. To do so, we can encode the non-termination of a 2CM (avoiding the $t$ counter). This further underlines the tightness of Theorem 4.6.

Lemma 4.13. *The satisfiability problem is undecidable for specifications $(\psi, \varphi)$ where $\varphi$ is of the form $\forall\exists^*.\Diamond\wedge\Diamond$.*

[3] The size of $Q_\psi$ is at most exponential in $\psi$ [42], so the size of the stack alphabet of $\mathcal{P}$ is at most double exponential in $\psi$. As deciding the emptiness of a Büchi pushdown system is possible in polynomial time, the 2EXPTIME upper bound follows.

*Example 4.14.* Using similar ideas as in Theorem 4.12, we can encode that a one-counter machine has an infinite computation with only a single eventuality (as we only need to ensure that the *single* counter is updated consistently). Combined with Theorem 4.6, we derive that we can decide the existence of an infinite computation in a one-counter machine. While this is long known (see, e.g., [29]), it nevertheless emphasizes that our newly identified decidable class is broader than it seems at first glance. ◁

## 4.6 Deterministic Liveness

Liveness for trace properties (cf. Definition 2.1) and hyperliveness (cf. Definition 2.3) already imply that a property is satisfiable. As demonstrated by Theorem 4.2, the same does not hold for temporal liveness hyperproperties. We can, however, identify a fragment within temporal liveness for which the intuition that liveness implies satisfiability transfers to the realm of hyperproperties. We say an LTL property $\phi$ is a *deterministic liveness* property if it is a liveness property and can be recognized by a *deterministic* Büchi automaton.

Proposition 4.15. *HyperLTL formulas of the form $\varphi = \forall\exists^*.\,\phi$ where $\phi$ is a deterministic liveness property are always satisfiable and have a finite model.*

Proof Sketch. In a deterministic automaton describing a liveness property, any reachable state has a path to an accepting state. We use this to iteratively construct a model. The full proof can be found in the full version [6]. □

Note that the same does not hold if we consider more than one universal quantifier. As an example, the formula $\forall\pi\forall\pi'.\,\Diamond(a_\pi \wedge \neg a_{\pi'})$ is unsatisfiable but $\Diamond(a_\pi \wedge \neg a_{\pi'})$ is a deterministic liveness property. If we consider deterministic liveness in combination with trace properties, we again obtain a jump to $\Sigma^1_1$-hardness.

Corollary 4.16. *Satisfiability is $\Sigma^1_1$-hard for specifications of the form $(\psi, \varphi)$ where $\varphi$ is of the form $\forall\exists^*.\,\phi$ and $\phi$ is a deterministic liveness property.*

Proof. Follows directly from Theorem 4.12 as conjunctions of eventualities are deterministic liveness. □

## 4.7 Overview: Liveness vs Safety

Our results provide a clear picture of the (un)decidability boundaries within fragments of HyperLTL. In particular, our systematic study allows a direct comparison between temporal safety and temporal liveness. For the full fragment, temporal liveness already subsumes satisfiability of full HyperLTL, which contrasts strongly with the much cheaper (albeit still undecidable) problem for temporal safety. This changes if we consider simpler fragments. Here, the $\Diamond$ fragment is drastically better behaved in terms of complexity and even admits large decidable fragments for cases where the safety counterpart already exhibits full analytical hardness.

# 5 FINDING LARGEST MODELS

To complement the decidability results from the previous sections, we propose a new (incomplete) algorithm to detect (un)satisfiability of $\forall\exists^*$ HyperLTL formulas. So far, the only available algorithm checks for finite models of bounded size and then iteratively increases the bound [21, 33]. Such an approach finds smallest models and cannot determine unsatisfiability. The key insight for our algorithm is that $\forall\exists^*$ formulas are closed under union, therefore, a formula $\varphi$ is satisfiable iff there is a (unique) *largest* model satisfying $\varphi$. To find such models algorithmically, we iteratively eliminate choices for the $\exists^*$ quantifiers that admit no witness trace when chosen for the $\forall$ quantifier. Thereby, we do not only find largest models but can also detect unsatisfiability. Our incremental elimination is closely related to a recent algorithm used in the context of finite-trace properties (which was developed independently) [10].

For presentation reasons, we present the algorithm for $\forall\exists$ formulas. Our implementation (see Section 6) supports full $\forall\exists^*$ properties.

**Algorithm 1: Algorithm that searches for the largest model of a $\forall\exists$ property. Initially, $\mathcal{A}$ is a Büchi automaton that accepts the body of the HyperLTL property.**

```
1: procedure findModel(A)
2:     if L(A^∀) = ∅ then
3:         return UNSAT;
4:     if L(A^∃) ⊆ L(A^∀) then
5:         return SAT, model: L(A^∀);
6:     A_new := A ∩ A^∀_{π'};
7:     findModel(A_new);
```

## 5.1 Algorithm

For a Büchi automaton $\mathcal{A}$ over $AP_\pi \cup AP_{\pi'}$, we define $\mathcal{A}^\forall$ and $\mathcal{A}^\exists$ as the automata (over $AP$) that (existentially) project $\mathcal{A}$ on the alphabet $AP_\pi$ and $AP_{\pi'}$, respectively. Let $\varphi = \forall\pi\exists\pi'.\,\phi$ be the HyperLTL formula for which we check satisfiability and let $\mathcal{A}_\phi$ be an automaton over $AP_\pi \cup AP_{\pi'}$ accepting $\phi$. In particular, $\mathcal{A}^\forall_\phi$ accepts all words for which there exists a witness trace for the existential quantifier. Our algorithm is depicted in Algorithm 1. Initially, we call findModel($\mathcal{A}_\phi$).

The first candidate is thus $\mathcal{A} = \mathcal{A}_\phi$. If $\mathcal{L}(\mathcal{A}^\forall) = \emptyset$, i.e., no trace has a witness trace in $\mathcal{A}$, $\varphi$ is unsatisfiable. If all potential witness traces in $\mathcal{L}(\mathcal{A}^\exists)$ are contained in $\mathcal{L}(\mathcal{A}^\forall)$ (so they have a witness trace themself), $\varphi$ is satisfiable and $\mathcal{L}(\mathcal{A}^\forall)$ is a model. If neither is the case, we refine $\mathcal{A}$ by removing all traces whose $\exists$ component is not in $\mathcal{L}(\mathcal{A}^\forall)$. We define $\mathcal{A}_{\text{new}}$ as the intersection $\mathcal{A} \cap \mathcal{A}^\forall_{\pi'}$, where $\mathcal{A}^\forall_{\pi'}$ is $\mathcal{A}^\forall$ with the alphabet changed from $AP$ to $AP_{\pi'}$. We can compute $\mathcal{A}_{\text{new}}$ via a standard intersection construction on Büchi automata. Automaton $\mathcal{A}_{\text{new}}$ might again contain witness traces that themselves have no witness trace, so we recurse.

## 5.2 Correctness

The algorithm maintains the following invariants.

Lemma 5.1. *In every iteration of Algorithm 1 it holds that $\mathcal{L}(\mathcal{A}_{new}) \subseteq \mathcal{L}(\mathcal{A})$, and for any trace set $T$ with $T \models \forall\pi\exists\pi'.\,\phi$, $T \subseteq \mathcal{L}(\mathcal{A}^\forall)$.*

Using Lemma 5.1 it is easy to see the following.

Proposition 5.2. *If Algorithm 1 terminates with* UNSAT*, then $\varphi$ is unsatisfiable. If it terminates with* SAT *and model $\mathcal{L}(\mathcal{A}^\forall)$, then $\mathcal{L}(\mathcal{A}^\forall)$ is the unique largest model of $\varphi$.*

**Table 2: Comparison of LMHyper and MGHyper on randomly generated formulas of varying size. We give the size of the formula's AST (Size), the percentage of solved formulas ($p$), the average time spent on solved cases in ms ($t$), and the average number of iterations (number of recursive calls) used by LMHyper (#Iter). The timeout is set to 5 sec per formula.**

| | MGHyper | | LMHyper | | |
|---|---|---|---|---|---|
| Size | $p$ | $t$ | $p$ | $t$ | #Iter |
| 15 | 95 % | 40 | 100 % | 235 | 0.38 |
| 16 | 93 % | 39 | 99 % | 239 | 0.44 |
| 17 | 95 % | 39 | 100 % | 221 | 0.43 |
| 18 | 92 % | 38 | 100 % | 201 | 0.39 |
| 19 | 95 % | 40 | 100 % | 180 | 0.43 |
| 20 | 97 % | 42 | 100 % | 215 | 0.27 |

To generalize to $\forall\exists^*$, we intersect the $\forall$-projection ($\mathcal{A}^\forall$) with each of the projections on existentially quantified positions. Models for $\forall^*\exists^*$-properties are, in general, not closed under union, so our algorithm does not extend beyond $\forall\exists^*$.

## 6 IMPLEMENTATION AND EXPERIMENTS

We have implemented the algorithm described in Section 5 in a tool called LMHyper (short for **L**argest **M**odel of **Hyper**LTL). LMHyper reads both a $\forall\exists^*$ HyperLTL formula $\varphi$ and LTL formula $\psi$ and searches for an (un)satisfiability proof for $(\psi, \varphi)$. Internally, we represent the current candidate as a generalized Büchi automaton and use spot [18] to perform automata operations. The only other available tool for $\forall\exists^*$ HyperLTL satisfiability is MGHyper [21], which implements the incremental approach to find models of bounded size.

### 6.1 Random Benchmarks

We compare LMHyper against MGHyper on random formulas where we sample the LTL body of a formula using randltl [18]. The results are given in Table 2. On our benchmarks, LMHyper usually takes longer than MGHyper but can handle a larger percentage of formulas. We observe that randomly generated HyperLTL formulas are, in most cases, satisfiable by a model with a single trace, as the atomic propositions are seldom shared between different trace variables. This explains the high success rate of MGHyper (see [21]) even though MGHyper cannot prove unsatisfiability.

### 6.2 Infinite and Large Models

We compiled a small number of more interesting properties that do not have single-trace models. Our results are depicted in Table 3. The Inf specification expresses that a model has infinitely many traces. Example 1.1 is the example from the introduction. The Enforce-$n$ specification enforces a model that has at least $n$ traces. It is defined as $\forall\pi\exists\pi_1 \dots \pi_n. \bigwedge_{i<j} \Diamond(a_{\pi_i} \not\leftrightarrow a_{\pi_j})$. The Unsat-$n$ specifications are unsatisfiable. Their definition is a trace property $\psi := (\neg a)\mathcal{U}(a \wedge \bigcirc\Box\neg a) \wedge \bigcirc^n \Box\neg a$ combined with the hyperproperty $\varphi := \forall\pi\exists\pi'. \Diamond(a_\pi \wedge \bigcirc a_{\pi'})$. The formula is designed such that Algorithm 1 requires $n$ iterations to discover unsatisfiability. MGHyper times out for most of the examples; even on simple properties like Enforce-3. In contrast, LMHyper can verify properties enforcing many traces in a single iteration because the number of iterations is independent of the number of traces in a model. As expected, Unsat-$n$ is unsatisfiable and LMHyper requires multiple iterations to show this.

**Table 3: Comparison of LMHyper and MGHyper on handcrafted specifications. We give the result (Res) (✓ if the specification is satisfiable and ✗ if it is unsatisfiable), the time in ms ($t$), and the number of iterations needed by LMHyper (#Iter). The timeout is set to 5 min.**

| | MGHyper | | LMHyper | | |
|---|---|---|---|---|---|
| Instance | Res | $t$ | Res | $t$ | #Iter |
| Inf | - | TO | ✓ | 350 | 1 |
| Example 1.1 | - | TO | ✓ | 232 | 1 |
| Enforce-2 | ✓ | 444 | ✓ | 262 | 0 |
| Enforce-3 | | TO | ✓ | 334 | 0 |
| Enforce-5 | - | TO | ✓ | 491 | 0 |
| Unsat-3 | - | TO | ✗ | 777 | 3 |
| Unsat-5 | - | TO | ✗ | 1363 | 5 |
| Unsat-9 | - | TO | ✗ | 1681 | 9 |

## 7 CONCLUSION

We have studied the satisfiability problem for $\forall^*\exists^*$ HyperLTL formulas in combination with LTL formulas describing functional behavior. To obtain results below the general $\Sigma^1_1$ complexity of HyperLTL, we have focused on simpler hyperproperties belonging to the classes of temporal safety and temporal liveness as well as fragments thereof. We have shown that temporal safety is an expressive class that is very well suited for satisfiability studies and enjoys coRE-completeness. In combination with general LTL properties, already very simple formulas like invariants cause $\Sigma^1_1$-completeness. The temporal liveness class, on the other hand, is $\Sigma^1_1$-complete in general but contains non-trivial fragments that are decidable, even in combination with arbitrary LTL formulas.

We have shown that functional specifications given in LTL play a significant role in the undecidability of general hyperproperties. The main open question for future work is whether further decidable fragments can be found by restricting the operator structure of the functional specification.

## ACKNOWLEDGMENTS

All authors are partially supported by the *German Research Foundation* (DFG) in project 389792660, TRR 248 (Center for Perspicuous Systems). M. Krötzsch is additionally supported by the *Bundesministerium für Bildung und Forschung* (BMBF) in project ScaDS.AI (Center for Scalable Data Analytics and Artificial Intelligence), and by the Center for Advancing Electronics Dresden (cfaed). R. Beutner and J. Hofmann carried out this work as members of the Saarbrücken Graduate School of Computer Science.

# REFERENCES


[1] Erika Ábrahám, Ezio Bartocci, Borzoo Bonakdarpour, and Oyendrila Dobe. 2020. Probabilistic Hyperproperties with Nondeterminism. In *International Symposium on Automated Technology for Verification and Analysis, ATVA 2020 (Lecture Notes in Computer Science, Vol. 12302)*. Springer. https://doi.org/10.1007/978-3-030-59152-6_29

[2] Bowen Alpern and Fred B. Schneider. 1985. Defining Liveness. *Inf. Process. Lett.* 21, 4 (1985). https://doi.org/10.1016/0020-0190(85)90056-0

[3] Rajeev Alur and Thomas A. Henzinger. 1994. A Really Temporal Logic. *J. ACM* 41, 1 (1994). https://doi.org/10.1145/174644.174651

[4] Ezio Bartocci, Thomas Ferrère, Thomas A. Henzinger, Dejan Nickovic, and Ana Oliveira da Costa. 2022. Flavors of Sequential Information Flow. In *International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2022 (Lecture Notes in Computer Science, Vol. 13182)*. Springer. https://doi.org/10.1007/978-3-030-94583-1_1

[5] Jan Baumeister, Norine Coenen, Borzoo Bonakdarpour, Bernd Finkbeiner, and César Sánchez. 2021. A Temporal Logic for Asynchronous Hyperproperties. In *International Conference on Computer Aided Verification, CAV 2021 (Lecture Notes in Computer Science, Vol. 12759)*. Springer. https://doi.org/10.1007/978-3-030-81685-8_33

[6] Raven Beutner, David Carral, Bernd Finkbeiner, Jana Hofmann, and Markus Krötzsch. 2022. Deciding Hyperproperties Combined with Functional Specifications. *CoRR* abs/2205.15138 (2022). arXiv:2205.15138

[7] Raven Beutner and Bernd Finkbeiner. 2021. A Temporal Logic for Strategic Hyperproperties. In *International Conference on Concurrency Theory, CONCUR 2021 (LIPIcs, Vol. 203)*. Schloss Dagstuhl. https://doi.org/10.4230/LIPIcs.CONCUR.2021.24

[8] Raven Beutner and Bernd Finkbeiner. 2022. Prophecy Variables for Hyperproperty Verification. In *IEEE Computer Security Foundations Symposium, CSF 2022*. IEEE.

[9] Raven Beutner and Bernd Finkbeiner. 2022. Software Verification of Hyperproperties Beyond $k$-Safety. In *International Conference on Computer Aided Verification, CAV 2022 (Lecture Notes in Computer Science)*. Springer.

[10] Borzoo Bonakdarpour and Sarai Sheinvald. 2022. Finite-Word Hyperlanguages. arXiv:2201.01670

[11] Ahmed Bouajjani, Javier Esparza, and Oded Maler. 1997. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *International Conference on Concurrency Theory, CONCUR 1997 (Lecture Notes in Computer Science, Vol. 1243)*. Springer. https://doi.org/10.1007/3-540-63141-0_10

[12] Laura Bozzelli, Adriano Peron, and César Sánchez. 2021. Asynchronous Extensions of HyperLTL. In *Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021*. IEEE. https://doi.org/10.1109/LICS52264.2021.9470583

[13] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. 2014. Temporal Logics for Hyperproperties. In *International Conference on Principles of Security and Trust, POST 2014 (Lecture Notes in Computer Science, Vol. 8414)*. Springer. https://doi.org/10.1007/978-3-642-54792-8_15

[14] Michael R. Clarkson and Fred B. Schneider. 2008. Hyperproperties. In *IEEE Computer Security Foundations Symposium, CSF 2008*. IEEE. https://doi.org/10.1109/CSF.2008.7

[15] Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. 2019. The Hierarchy of Hyperlogics. In *Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019*. IEEE. https://doi.org/10.1109/LICS.2019.8785713

[16] Pedro R. D'Argenio, Gilles Barthe, Sebastian Biewer, Bernd Finkbeiner, and Holger Hermanns. 2017. Is Your Software on Dope? - Formal Analysis of Surreptitiously "enhanced" Programs. In *European Symposium on Programming, ESOP 2017 (Lecture Notes in Computer Science, Vol. 10201)*. Springer. https://doi.org/10.1007/978-3-662-54434-1_4

[17] Rayna Dimitrova, Bernd Finkbeiner, and Hazem Torfah. 2020. Probabilistic Hyperproperties of Markov Decision Processes. In *International Symposium on Automated Technology for Verification and Analysis, ATVA 2020 (Lecture Notes in Computer Science, Vol. 12302)*. Springer. https://doi.org/10.1007/978-3-030-59152-6_27

[18] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. 2016. Spot 2.0 - A Framework for LTL and $\omega$-Automata Manipulation. In *International Symposium on Automated Technology for Verification and Analysis, ATVA 2016 (Lecture Notes in Computer Science, Vol. 9938)*. https://doi.org/10.1007/978-3-319-46520-3_8

[19] Bernd Finkbeiner, Lennart Haas, and Hazem Torfah. 2019. Canonical Representations of k-Safety Hyperproperties. In *IEEE Computer Security Foundations Symposium, CSF 2019*. IEEE. https://doi.org/10.1109/CSF.2019.00009

[20] Bernd Finkbeiner and Christopher Hahn. 2016. Deciding Hyperproperties. In *International Conference on Concurrency Theory, CONCUR 2016 (LIPIcs, Vol. 59)*. Schloss Dagstuhl. https://doi.org/10.4230/LIPIcs.CONCUR.2016.13

[21] Bernd Finkbeiner, Christopher Hahn, and Tobias Hans. 2018. MGHyper: Checking Satisfiability of HyperLTL Formulas Beyond the $\exists^*\forall^*$ Fragment. In *International Symposium on Automated Technology for Verification and Analysis, ATVA 2018 (Lecture Notes in Computer Science, Vol. 11138)*. Springer. https://doi.org/10.1007/978-3-030-01090-4_31

[22] Bernd Finkbeiner, Christopher Hahn, and Hazem Torfah. 2018. Model Checking Quantitative Hyperproperties. In *International Conference on Computer Aided Verification, CAV 2018 (Lecture Notes in Computer Science, Vol. 10981)*. Springer. https://doi.org/10.1007/978-3-319-96145-3_8

[23] Bernd Finkbeiner and Martin Zimmermann. 2017. The First-Order Logic of Hyperproperties. In *Symposium on Theoretical Aspects of Computer Science, STACS 2017 (LIPIcs, Vol. 66)*. Schloss Dagstuhl. https://doi.org/10.4230/LIPIcs.STACS.2017.30

[24] Michael J. Fischer and Richard E. Ladner. 1979. Propositional Dynamic Logic of Regular Programs. *J. Comput. Syst. Sci.* 18, 2 (1979). https://doi.org/10.1016/0022-0000(79)90046-1

[25] Marie Fortin, Louwe B. Kuijer, Patrick Totzke, and Martin Zimmermann. 2021. HyperLTL Satisfiability is $\Sigma^1_1$-complete, HyperCTL* Satisfiability is $\Sigma^2_1$-complete. In *International Symposium on Mathematical Foundations of Computer Science, MFCS 2021 (LIPIcs, Vol. 202)*. Schloss Dagstuhl. https://doi.org/10.4230/LIPIcs.MFCS.2021.47

[26] Kurt Gödel. 1929. *Über die vollständigkeit des logikkalküls.*

[27] Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. 2020. Propositional Dynamic Logic for Hyperproperties. In *International Conference on Concurrency Theory, CONCUR 2020 (LIPIcs, Vol. 171)*. Schloss Dagstuhl. https://doi.org/10.4230/LIPIcs.CONCUR.2020.50

[28] Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. 2021. Automata and fixpoints for asynchronous hyperproperties. *Proc. ACM Program. Lang.* 5, POPL (2021). https://doi.org/10.1145/3434319

[29] Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. 2009. Reachability in Succinct and Parametric One-Counter Automata. In *International Conference on Concurrency Theory, CONCUR 2009 (Lecture Notes in Computer Science, Vol. 5710)*. Springer. https://doi.org/10.1007/978-3-642-04081-8_25

[30] Andreas Krebs, Arne Meier, Jonni Virtema, and Martin Zimmermann. 2018. Team Semantics for the Specification and Verification of Hyperproperties. In *International Symposium on Mathematical Foundations of Computer Science, MFCS 2018 (LIPIcs, Vol. 117)*. Schloss Dagstuhl. https://doi.org/10.4230/LIPIcs.MFCS.2018.10

[31] Orna Kupferman and Moshe Y. Vardi. 1999. Model Checking of Safety Properties. In *International Conference on Computer Aided Verification, CAV 1999 (Lecture Notes in Computer Science, Vol. 1633)*. Springer. https://doi.org/10.1007/3-540-48683-6_17

[32] Martin Lück. 2016. Complete Problems of Propositional Logic for the Exponential Hierarchy. *CoRR* abs/1602.03050 (2016). arXiv:1602.03050

[33] Corto Mascle and Martin Zimmermann. 2020. The Keys to Decidable HyperLTL Satisfiability: Small Models or Very Simple Formulas. In *EACSL Annual Conference on Computer Science Logic, CSL 2020 (LIPIcs, Vol. 152)*. Schloss Dagstuhl. https://doi.org/10.4230/LIPIcs.CSL.2020.29

[34] Daryl McCullough. 1988. Noninterference and the composability of security properties. In *IEEE Symposium on Security and Privacy, Oakland, SP 1988*. IEEE. https://doi.org/10.1109/SECPRI.1988.8110

[35] Marvin Lee Minsky. 1967. *Computation.* Prentice-Hall Englewood Cliffs.

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

[37] John Alan Robinson and Andrei Voronkov (Eds.). 2001. *Handbook of Automated Reasoning.* MIT Press.

[38] Hartley Rogers Jr. 1987. *Theory of recursive functions and effective computability.* MIT press.

[39] A Prasad Sistla. 1994. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing* 6, 5 (1994).

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

[41] Larry J. Stockmeyer. 1976. The Polynomial-Time Hierarchy. *Theor. Comput. Sci.* 3, 1 (1976). https://doi.org/10.1016/0304-3975(76)90061-X

[42] Moshe Y. Vardi and Pierre Wolper. 1994. Reasoning About Infinite Computations. *Inf. Comput.* 115, 1 (1994). https://doi.org/10.1006/inco.1994.1092

[43] Jonni Virtema, Jana Hofmann, Bernd Finkbeiner, Juha Kontinen, and Fan Yang. 2021. Linear-Time Temporal Logic with Team Semantics: Expressivity and Complexity. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021 (LIPIcs, Vol. 213)*. Schloss Dagstuhl. https://doi.org/10.4230/LIPIcs.FSTTCS.2021.52