
# Verifying Hyperliveness

Norine Coenen[1(✉)], Bernd Finkbeiner[1], César Sánchez[2], and Leander Tentrup[1]

[1] Reactive Systems Group, Saarland University, Saarbrücken, Germany
coenen@react.uni-saarland.de
[2] IMDEA Software Institute, Madrid, Spain


**Abstract.** HyperLTL is an extension of linear-time temporal logic for the specification of hyperproperties, i.e., temporal properties that relate multiple computation traces. HyperLTL can express information flow policies as well as properties like symmetry in mutual exclusion algorithms or Hamming distances in error-resistant transmission protocols. Previous work on HyperLTL model checking has focussed on the alternation-free fragment of HyperLTL, where verification reduces to checking a standard trace property over an appropriate self-composition of the system. The alternation-free fragment does, however, not cover general hyperliveness properties. Universal formulas, for example, cannot express the secrecy requirement that for every possible value of a secret variable there exists a computation where the value is different while the observations made by the external observer are the same. In this paper, we study the more difficult case of hyperliveness properties expressed as HyperLTL formulas with quantifier alternation. We reduce existential quantification to strategic choice and show that synthesis algorithms can be used to eliminate the existential quantifiers automatically. We furthermore show that this approach can be extended to reactive system synthesis, i.e., to automatically construct a reactive system that is guaranteed to satisfy a given HyperLTL formula.

## 1 Introduction

HyperLTL [6] is a temporal logic for *hyperproperties* [7], i.e., for properties that relate multiple computation traces. Hyperproperties cannot be expressed in standard linear-time temporal logic (LTL), because LTL can only express *trace properties*, i.e., properties that characterize the correctness of individual computations. Even branching-time temporal logics like CTL and CTL*, which quantify

This work was partially supported by the German Research Foundation (DFG) as part of the Collaborative Research Center "Foundations of Perspicuous Software Systems" (TRR 248, 389792660), and by the European Research Council (ERC) Grant OSARES (No. 683300)., by Madrid Reg. Government project "S2018/TCS-4339 (BLOQUES-CM)", by EU H2020 project 731535 "Elastest" and by Spanish National Project "BOSCO (PGC2018-102210-B-100)".


I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 121–139, 2019.
https://doi.org/10.1007/978-3-030-25540-4_7

over computation paths, cannot express hyperproperties, because quantifying over a second path automatically means that the subformula can no longer refer to the previously quantified path. HyperLTL addresses this limitation with quantifiers over trace variables, which allow the subformula to refer to all previously chosen traces. For example, *noninterference* [21] between a secret input $h$ and a public output $o$ can be specified in HyperLTL by requiring that all pairs of traces $\pi$ and $\pi'$ that always have the same inputs except for $h$ (i.e., all inputs in $I \setminus \{h\}$ are equal on $\pi$ and $\pi'$) also have the same output $o$ at all times:

$$\forall \pi . \forall \pi' .\ \square \Big( \bigwedge_{i \in I \setminus \{h\}} i_\pi = i_{\pi'} \Big) \ \Rightarrow\ \square (o_\pi = o_{\pi'})$$

This formula states that a change in the secret input $h$ alone cannot cause any difference in the output $o$.

For certain properties of interest, the additional expressiveness of HyperLTL comes at no extra cost when considering the model checking problem. To check a property like noninterference, which only has universal trace quantifiers, one simply builds the self-composition of the system, which provides a separate copy of the state variables for each trace. Instead of quantifying over all pairs of traces, it then suffices to quantify over individual traces of the self-composed system, which can be done with standard LTL. Model checking universal formulas is NLOGSPACE-complete in the size of the system and PSPACE-complete in the size of the formula, which is precisely the same complexity as for LTL.

Universal HyperLTL formulas suffice to express hypersafety properties like noninterference, but not hyperliveness properties that require, in general, quantifier alternation. A prominent example is *generalized noninterference* (GNI) [27], which can be expressed as the following HyperLTL formula:

$$\forall \pi . \forall \pi' . \exists \pi'' .\ \square (h_\pi = h_{\pi''}) \ \wedge\ \square (o_{\pi'} = o_{\pi''})$$

This formula requires that for every pair of traces $\pi$ and $\pi'$, there is a third trace $\pi''$ in the system that agrees with $\pi$ on $h$ and with $\pi'$ on $o$. The existence of an appropriate trace $\pi''$ ensures that in $\pi$ and $\pi'$, the value of $o$ is not determined by the value of $h$. Generalized noninterference stipulates that low-security outputs may not be altered by the injection of high-security inputs, while permitting nondeterminism in the low-observable behavior. The existential quantifier is needed to allow this nondeterminism. GNI is a hyperliveness property [7] even though the underlying LTL formula is a safety property. The reason for that is that we can extend any set of traces that violates GNI into a set of traces that satisfies GNI, by adding, for each offending pair of traces $\pi, \pi'$, an appropriate trace $\pi''$.

Hyperliveness properties also play an important role in applications beyond security. For example, *robust cleanness* [9] specifies that significant differences in the output behavior are only permitted after significant differences in the input:

$$\forall \pi . \forall \pi' . \exists \pi'' .\ \square \big(i_{\pi'} = i_{\pi''}\big) \ \wedge\ \big(\hat{d}(o_\pi, o_{\pi''}) \leq \kappa_o \ \mathcal{W}\ \hat{d}(i_\pi, i_{\pi''}) > \kappa_i\big)$$

The differences are measured by a distance function $\hat{d}$ and compared to constant thresholds $\kappa_i$ for the input and $\kappa_o$ for the output. The formula specifies

the existence of a trace $\pi''$ that globally agrees with $\pi'$ on the input and where the difference in the output $o$ between $\pi$ and $\pi''$ is bounded by $\kappa_o$, unless the difference in the input $i$ between $\pi$ and $\pi''$ was greater than $\kappa_i$. Robust cleanness, thus, forbids unexpected jumps in the system behavior that are, for example, due to software doping, while allowing for behavioral differences due to nondeterminism.

With quantifier alternation, the model checking problem becomes much more difficult. Model checking HyperLTL formulas of the form $\forall^*\exists^*\varphi$, where $\varphi$ is a quantifier-free formula, is PSPACE-complete in the size of the system and EXPSPACE-complete in the formula. The only known model checking algorithm replaces the existential quantifier with the negation of a universal quantifier over the negated subformula; but this requires a complementation of the system behavior, which is completely impractical for realistic systems.

In this paper, we present an alternative approach to the verification of hyperliveness properties. We view the model checking problem of a formula of the form $\forall\pi.\exists\pi'.\ \varphi$ as a game between the $\forall$-player and the $\exists$-player. While the $\forall$-player moves through the state space of the system building trace $\pi$, the $\exists$-player must match each move in a separate traversal of the state space resulting in a trace $\pi'$ such that the pair $\pi, \pi'$ satisfies $\varphi$. Clearly, the existence of a winning strategy for the $\exists$-player implies that $\forall\pi.\exists\pi'.\ \varphi$ is satisfied. The converse is not necessarily true: Even if there always is a trace $\pi'$ that matches the universally chosen trace $\pi$, the $\exists$-player may not be able to construct this trace, because she only knows about the choices made by the $\forall$-player in the finite prefix of $\pi$ that has occurred so far, and not the choices that will be made by the $\forall$-player in the infinite future. We address this problem by introducing *prophecy variables* into the system. Without changing the behavior of the system, the prophecy variables give the $\exists$-player the information about the future that is needed to make the right choice after seeing only the finite prefix. Such prophecy variables can be provided manually by the user of the model checker to provide a lookahead on future moves of the $\forall$-player.

This game-theoretic approach provides an opportunity for the user to reduce the complexity of the model checking problem: If the user provides a strategy for the $\exists$-player, then the problem reduces to the cheaper model checking problem for universal properties. We show that such strategies can also be constructed automatically using synthesis. Beyond model checking, the game-theoretic approach also provides a method for the synthesis of systems that satisfy a conjunction of hypersafety and hyperliveness properties. Here, we do not only synthesize the strategy, but also construct the system itself, i.e., the game graph on which the model checking game is played. While the synthesis from $\forall^*\exists^*$ hyperproperties is known to be undecidable in general, we show that the game-theoretic approach can naturally be integrated into bounded synthesis, which checks for the existence of a correct system up to a bound on the number of states.

**Related Work.** While the verification of general HyperLTL formulas has been studied before [6,17,18], there has been, so far, no practical model checking algorithm for HyperLTL formulas with quantifier alternation. The existing algorithm involves a complementation of the system automaton, which results in an

exponential blow-up of the state space [18]. The only existing model checker for HyperLTL, MCHyper [18], was therefore, so far, limited to the alternation-free fragment. Although some hyperliveness properties lie in this fragment, quantifier alternation is needed to express general hyperliveness properties like GNI. In this paper, we present a technique to model check these hyperliveness properties and extend MCHyper to formulas with quantifier alternation.

The situation is similar in the area of reactive synthesis. There is a synthesis algorithm that automatically constructs implementations from HyperLTL specifications [13] using the bounded synthesis approach [20]. This algorithm is, however, also only applicable to the alternation-free fragment of HyperLTL. In this paper, we extend the bounded synthesis approach to HyperLTL formulas with quantifier alternation. Beyond the model checking and synthesis problems, the satisfiability [11,12,14] and monitoring [15,16,22] problems of HyperLTL have also been studied in the past.

For certain information-flow security policies, there are verification techniques that use methods related to our model checking and synthesis algorithms. Specifically, the self-composition technique [2,3], a construction based on the product of copies of a system, has been tailored for various trace-based security definitions [10,23,28]. Unlike our algorithms, these techniques focus on specific information-flow policies, not on a general logic like HyperLTL.

The use of prophecy variables [1] to make information about the future accessible is a known technique in the verification of trace properties. It is, for example, used to establish simulation relations between automata [26] or in the verification of CTL$^*$ properties [8].

In our game-theoretic view on the model checking problem for $\forall^*\exists^*$ hyperproperties the $\exists$-player has an infinite lookahead. There is some work on *finite* lookahead on trace languages [24]. We use the idea of finite lookahead as an approximation to construct existential strategies and give a novel synthesis construction for strategies with delay based on bounded synthesis [20].

## 2 Preliminaries

For tuples $\boldsymbol{x} \in X^n$ and $\boldsymbol{y} \in X^m$ over set $X$, we use $\boldsymbol{x} \cdot \boldsymbol{y} \in X^{n+m}$ to denote the concatenation of $\boldsymbol{x}$ and $\boldsymbol{y}$. Given a function $f\colon X \to Y$ and a tuple $\boldsymbol{x} \in X^n$, we define by $f \circ \boldsymbol{x} \in Y^n$ the tuple $(f(\boldsymbol{x}[1]), \dots, f(\boldsymbol{x}[n]))$. Let AP be a finite set of atomic propositions and let $\Sigma = 2^{\text{AP}}$ be the corresponding alphabet. A *trace* $t \in \Sigma^\omega$ is an infinite sequence of elements of $\Sigma$. We denote a set of traces by $Tr \subseteq \Sigma^\omega$. We define $t[i, \infty]$ to be the suffix of $t$ starting at position $i \geq 0$.

*HyperLTL.* HyperLTL [6] is a temporal logic for specifying hyperproperties. It extends LTL by quantification over trace variables $\pi$ and a method to link atomic propositions to specific traces. Let $\mathcal{V}$ be an infinite set of trace variables. Formulas in HyperLTL are given by the grammar

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

where $a \in \text{AP}$ and $\pi \in \mathcal{V}$. We allow the standard boolean connectives $\wedge$, $\rightarrow$, $\leftrightarrow$ as well as the derived LTL operators release $\varphi \mathcal{R} \psi \equiv \neg(\neg\varphi \mathcal{U} \neg\psi)$, eventually $\Diamond\varphi \equiv \mathit{true}\ \mathcal{U}\ \varphi$, globally $\Box\varphi \equiv \neg\Diamond\neg\varphi$, and weak until $\varphi \mathcal{W} \psi \equiv \Box\varphi \vee (\varphi\ \mathcal{U}\ \psi)$.

We call a $\mathcal{Q}^+\mathcal{Q}'^+\varphi$ HyperLTL formula (for $\mathcal{Q}, \mathcal{Q}' \in \{\forall, \exists\}$ and quantifier-free formula $\varphi$) *alternation-free* iff $\mathcal{Q} = \mathcal{Q}'$. Further, we say that $\mathcal{Q}^+\mathcal{Q}'^+\varphi$ has *one quantifier alternation* (or lies in the *one-alternation fragment*) iff $\mathcal{Q} \neq \mathcal{Q}'$.

The semantics of HyperLTL is given by the satisfaction relation $\vDash_{Tr}$ over a set of traces $Tr \subseteq \Sigma^\omega$. We define an assignment $\Pi : \mathcal{V} \to \Sigma^\omega$ that maps trace variables to traces. $\Pi[\pi \mapsto t]$ updates $\Pi$ by assigning variable $\pi$ to trace $t$.

$$
\begin{array}{ll}
\Pi, i \vDash_{Tr} a_\pi & \text{iff } a \in \Pi(\pi)[i] \\
\Pi, i \vDash_{Tr} \neg\varphi & \text{iff } \Pi, i \nvDash_{Tr} \varphi \\
\Pi, i \vDash_{Tr} \varphi \vee \psi & \text{iff } \Pi, i \vDash_{Tr} \varphi \text{ or } \Pi, i \vDash_{Tr} \psi \\
\Pi, i \vDash_{Tr} \bigcirc\varphi & \text{iff } \Pi, i+1 \vDash_{Tr} \varphi \\
\Pi, i \vDash_{Tr} \varphi\,\mathcal{U}\,\psi & \text{iff } \exists j \geq i.\, \Pi, j \vDash_{Tr} \psi \wedge \forall i \leq k < j.\, \Pi, k \vDash_{Tr} \varphi \\
\Pi, i \vDash_{Tr} \exists\pi.\,\varphi & \text{iff there is some } t \in Tr \text{ such that } \Pi[\pi \mapsto t], i \vDash_{Tr} \varphi \\
\Pi, i \vDash_{Tr} \forall\pi.\,\varphi & \text{iff for all } t \in Tr \text{ it holds that } \Pi[\pi \mapsto t], i \vDash_{Tr} \varphi
\end{array}
$$

We write $Tr \vDash \varphi$ for $\{\}, 0 \vDash_{Tr} \varphi$ where $\{\}$ denotes the empty assignment.

Every hyperproperty is an intersection of a hypersafety and a hyperliveness property [7]. A *hypersafety* property is one where there is a finite set of finite traces that is a bad prefix, i.e., that cannot be extended into a set of traces that satisfies the hypersafety property. A *hyperliveness* property is a property where every finite set of finite traces can be extended to a possibly infinite set of infinite traces such that the resulting trace set satisfies the hyperliveness property.

*Transition Systems.* We use transition systems as a model of computation for reactive systems. Transition systems consume sequences over an input alphabet by transforming their internal state in every step. Let $I$ and $O$ be a finite set of input and output propositions, respectively, and let $\Upsilon = 2^I$ and $\Gamma = 2^O$ be the corresponding finite alphabets. A $\Gamma$-labeled $\Upsilon$-*transition system* $\mathcal{S}$ is a tuple $\langle S, s_0, \tau, l\rangle$, where $S$ is a finite set of states, $s_0 \in S$ is the designated initial state, $\tau\colon S \times \Upsilon \to S$ is the transition function, and $l\colon S \to \Gamma$ is the state-labeling function. We write $s \xrightarrow{\upsilon} s'$ or $(s, \upsilon, s') \in \tau$ if $\tau(s, \upsilon) = s'$. We generalize the transition function to sequences over $\Upsilon$ by defining $\tau^*\colon \Upsilon^* \to S$ recursively as $\tau^*(\epsilon) = s_0$ and $\tau^*(\upsilon_0 \cdots \upsilon_{n-1}\upsilon_n) = \tau(\tau^*(\upsilon_0 \cdots \upsilon_{n-1}), \upsilon_n)$ for $\upsilon_0 \cdots \upsilon_{n-1}\upsilon_n \in \Upsilon^+$. Given an infinite word $\upsilon = \upsilon_0\upsilon_1 \ldots \in \Upsilon^\omega$, the transition system produces an infinite sequence of outputs $\gamma = \gamma_0\gamma_1\gamma_2 \ldots \in \Gamma^\omega$, such that $\gamma_i = l(\tau^*(\upsilon_0 \ldots \upsilon_{i-1}))$ for every $i \geq 0$. The resulting *trace* $\rho$ is $(\upsilon_0 \cup \gamma_0)(\upsilon_1 \cup \gamma_1) \ldots \in \Sigma^\omega$ where we have $AP = I \cup O$. The set of traces generated by $\mathcal{S}$ is denoted by $\mathit{traces}(\mathcal{S})$. Furthermore, we define $\varepsilon = \langle \{s\}, s, \tau_\varepsilon, l_\varepsilon\rangle$ as the transition system over $I = O = \emptyset$ that has only a single trace, that is $\mathit{traces}(\varepsilon) = \{\emptyset^\omega\}$. For this transition system, $\tau_\varepsilon(s, \emptyset) = s$ and $l_\varepsilon(s) = \emptyset$. Given two transition systems $\mathcal{S} = \langle S, s_0, \tau, l\rangle$ and $\mathcal{S}' = \langle S', s_0', \tau', l'\rangle$, we define $\mathcal{S} \times \mathcal{S}' = \langle S \times S', (s_0, s_0'), \tau'', l''\rangle$ as the $\Gamma^2$-labeled $\Upsilon^2$-transition system where $\tau''((s, s'), (\upsilon, \upsilon')) = (\tau(s, \upsilon), \tau'(s', \upsilon'))$ and $l''((s, s')) = (l(s), l'(s'))$. A transition system $\mathcal{S}$ satisfies a general HyperLTL formula $\varphi$, if, and only if, $\mathit{traces}(\mathcal{S}) \vDash \varphi$.

*Automata.* An alternating parity automaton $\mathcal{A}$ over a finite alphabet $\Sigma$ is a tuple $\langle Q, q_0, \delta, \alpha \rangle$, where $Q$ is a finite set of states, $q_0 \in Q$ is the designated initial state, $\delta \colon Q \times \Sigma \to \mathbb{B}^+(Q)$ is the transition function, and $\alpha \colon Q \to C$ is a function that maps states of $\mathcal{A}$ to a finite set of colors $C \subset \mathbb{N}$. For $C = \{0, 1\}$ and $C = \{1, 2\}$, we call $\mathcal{A}$ a co-Büchi and Büchi automaton, respectively, and we use the sets $F \subseteq Q$ and $B \subseteq Q$ to represent the rejecting ($C = 1$) and accepting ($C = 2$) states in the respective automaton (as a replacement of the coloring function $\alpha$). A safety automaton is a Büchi automaton where every state is accepting. The transition function $\delta$ maps a state $q \in Q$ and some $a \in \Sigma$ to a positive Boolean combination of successor states $\delta(q, a)$. An automaton is *non-deterministic* or *universal* if $\delta$ is purely disjunctive or conjunctive, respectively.

A run of an alternating automaton is a $Q$-labeled tree. A tree $T$ is a subset of $\mathbb{N}_{>0}^*$ such that for every node $n \in \mathbb{N}_{>0}^*$ and every positive integer $i \in \mathbb{N}_{>0}$, if $n \cdot i \in T$ then (i) $n \in T$ (i.e., $T$ is prefix-closed), and (ii) for every $0 < j < i$, $n \cdot j \in T$. The root of $T$ is the empty sequence $\epsilon$ and for a node $n \in T$, $|n|$ is the length of the sequence $n$, in other words, its distance from the root. A run of $\mathcal{A}$ on an infinite word $\rho \in \Sigma^\omega$ is a $Q$-labeled tree $(T, r)$ such that $r(\epsilon) = q_0$ and for every node $n \in T$ with children $n_1, \ldots, n_k$ the following holds: $1 \leq k \leq |Q|$ and $\{r(n_1), \ldots, r(n_k)\} \vDash \delta(q, \rho[i])$, where $q = r(n)$ and $i = |n|$. A path is accepting if the highest color appearing infinitely often is even. A run is accepting if all its paths are accepting. The language of $\mathcal{A}$, written $\mathcal{L}(\mathcal{A})$, is the set $\{\rho \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \rho\}$. A transition system $\mathcal{S}$ is accepted by an automaton $\mathcal{A}$, written $\mathcal{S} \vDash \mathcal{A}$, if $\mathit{traces}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{A})$.

*Strategies.* Given two disjoint finite alphabets $\Upsilon$ and $\Gamma$, a strategy $\sigma \colon \Upsilon^* \to \Gamma$ is a mapping from finite histories of $\Upsilon$ to $\Gamma$. A transition system $\mathcal{S} = \langle S, s_0, \tau, l \rangle$ *generates* the strategy $\sigma$ if $\sigma(\boldsymbol{\upsilon}) = l(\tau^*(\boldsymbol{\upsilon}))$ for every $\boldsymbol{\upsilon} \in \Upsilon^*$. A strategy $\sigma$ is called *finite-state* if there exists a transition system that generates $\sigma$.

In the following, we use finite-state strategies to modify the inputs of transition systems. Let $\mathcal{S} = \langle S, s_0, \tau, l \rangle$ be a transition system over input and output alphabets $\Upsilon$ and $\Gamma$ and let $\sigma \colon (\Upsilon')^* \to \Upsilon$ be a finite-state strategy. Let $\mathcal{S}' = \langle S', s_0', \tau', l' \rangle$ be the transition system implementing $\sigma$, then $\mathcal{S} \mid\mid \sigma = \mathcal{S} \mid\mid \mathcal{S}'$ is the transition system $\langle S \times S', (s_0, s_0'), \tau^{||}, l^{||} \rangle$ where $\tau^{||} \colon (S \times S') \times \Upsilon' \to (S \times S')$ is defined as $\tau^{||}((s, s'), \upsilon') = (\tau(s, l'(s')), \tau'(s', \upsilon'))$ and $l^{||} \colon (S \times S') \to \Gamma$ is defined as $l^{||}(s, s') = l(s)$ for every $s \in S$, $s' \in S'$, and $\upsilon' \in \Upsilon'$.

*Model Checking HyperLTL.* We recap the model checking of universal HyperLTL formulas. This case, as well as the dual case of only existential quantifiers, is well-understood and, in fact, efficiently implemented in the model checker MCHyper [18]. The principle behind the model checking approach is *self-composition*, where we check a standard trace property on a composition of an appropriate number of copies of the given system.

Let *zip* denote the function that maps an $n$-tuple of sequences to a single sequence of $n$-tuples, for example, $\mathit{zip}([1, 2, 3], [4, 5, 6]) = [(1, 4), (2, 5), (3, 6)]$, and let *unzip* denote its inverse. Given $\mathcal{S} = \langle S, s_0, \tau, l \rangle$, the $n$-fold self-composition of $\mathcal{S}$ is the transition system $\mathcal{S}^n = \langle S^n, \boldsymbol{s_0'}, \tau_n, l_n \rangle$, where $\boldsymbol{s_0'} \coloneqq (s_0, \ldots, s_0) \in S^n$, $\tau_n(\boldsymbol{s}, \boldsymbol{\upsilon}) \coloneqq \tau \circ \mathit{zip}(\boldsymbol{s}, \boldsymbol{\upsilon})$ and $l_n(\boldsymbol{s}) \coloneqq l \circ \boldsymbol{s}$ for every $\boldsymbol{s} \in S^n$ and $\boldsymbol{\upsilon} \in \Upsilon^n$. If $\mathit{traces}(\mathcal{S})$

is the set of traces generated by $\mathcal{S}$, then $\{zip(\rho_1, \ldots, \rho_n) \mid \rho_1, \ldots, \rho_n \in traces(\mathcal{S})\}$ is the set of traces generated by $\mathcal{S}^n$. We use the notation $zip(\varphi, \pi_1, \pi_2, \ldots, \pi_n)$ for some HyperLTL formula $\varphi$ to combine the trace variables $\pi_1, \pi_2, \ldots, \pi_n$ (occurring free in $\varphi$) into a fresh trace variable $\pi^*$.

**Theorem 1 (Self-composition for universal HyperLTL formulas [18]).** *For a transition system $\mathcal{S}$ and a HyperLTL formula of the form $\forall\pi_1.\ \forall\pi_2. \ldots \forall\pi_n.\ \varphi$ it holds that $\mathcal{S} \vDash \forall\pi_1.\forall\pi_2. \ldots \forall\pi_n.\ \varphi$ iff $\mathcal{S}^n \vDash \forall\pi^*.\ zip(\varphi, \pi_1, \pi_2, \ldots, \pi_n)$.*

**Theorem 2 (Complexity of model checking universal formulas [18]).** *The model checking problem for universal HyperLTL formulas is PSPACE-complete in the size of the formula and NLOGSPACE-complete in the size of the transition system.*

The complexity of verifying universal HyperLTL formulas is exactly the same as the complexity of verifying LTL formulas. For HyperLTL formulas with quantifier alternations, the model checking problem is significantly more difficult.

**Theorem 3 (Complexity of model checking formulas with one quantifier alternation [18]).** *The model checking problem for HyperLTL formulas with one quantifier alternation is in EXPSPACE in the size of the formula and in PSPACE in the size of the transition system.*

One way to circumvent this complexity is to fix the existential choice and strengthen the formula to the universal fragment [9,13,18]. While avoiding the complexity problem, this transformation requires deep knowledge of the system, is prone to errors, and cannot be verified automatically as the problem of checking implications becomes undecidable [11]. In the following section, we present a technique that circumvents the complexity problem while still inheriting strong correctness guarantees. Further, we provide a method that can, under certain restrictions, derive a strategy for the existential choice automatically.

## 3 Model Checking with Quantifier Alternations

### 3.1 Model Checking with Given Strategies

Our first goal is the verification of HyperLTL formulas with one quantifier alternation, i.e., formulas of the form $\forall^*\exists^*\varphi$ or $\exists^*\forall^*\varphi$, where $\varphi$ is a quantifier-free formula. Note that the presented techniques can, similar to skolemization, be extended to more than one quantifier alternation. Quantifier alternation introduces dependencies between the quantified traces. In a $\forall^*\exists^*\varphi$ formula, the choices of the existential quantifiers depend on the choices of the universal quantifiers preceding them. In a formula of the form $\exists^*\forall^*\varphi$, however, there has to be a single choice for the existential quantifiers that works for all choices of the universal quantifiers. In this case, the existentially quantified variables do not depend on the universally quantified variables. Hence, the witnesses for the existential quantifiers are traces rather than functions that map tuples of traces

to traces. As established above, the model checking problem for HyperLTL formulas with quantifier alternation is known to be significantly more difficult than the model checking problem for universal formulas.

Our verification technique for formulas with quantifier alternation is to substitute strategic choice for existential choice. As discussed in the introduction, the existence of a strategy implies the existence of a trace.

**Theorem 4 (Substituting Strategic Choice for Existential Choice).** *Let $\mathcal{S}$ be a transition system over input alphabet $\Upsilon$.*
*It holds that $\mathcal{S} \vDash \forall\pi_1\forall\pi_2 \dots \forall\pi_n.\ \exists\pi'_1\exists\pi'_2 \dots \exists\pi'_m.\ \varphi$ if there is a strategy $\sigma : (\Upsilon^n)^* \to \Upsilon^m$ such that $\mathcal{S}^n \times (\mathcal{S}^m \mid\mid \sigma) \vDash \forall\pi^*.zip(\varphi, \pi_1, \pi_2, \dots \pi_n, \pi'_1, \pi'_2, \dots, \pi'_m)$.*
*It holds that $\mathcal{S} \vDash \exists\pi_1\exists\pi_2 \dots \exists\pi_m.\ \forall\pi'_1\forall\pi'_2 \dots \forall\pi'_n.\ \varphi$ if there is a strategy $\sigma : (\Upsilon^0)^* \to \Upsilon^m$ such that $(\mathcal{S}^m \mid\mid \sigma) \times \mathcal{S}^n \vDash \forall\pi^*.zip(\varphi, \pi_1, \pi_2, \dots \pi_m, \pi'_1, \pi'_2, \dots, \pi'_n)$.*

*Proof.* Let $\sigma$ be such a strategy, then we define a witness for the existential trace quantifiers $\exists\pi'_1\exists\pi'_2 \dots \exists\pi'_m$ as the sequence of inputs $\upsilon = \upsilon_0\upsilon_1 \dots \in (\Upsilon^m)^\omega$ such that $\upsilon_i = \sigma(\upsilon'_0\upsilon'_1 \dots \upsilon'_{i-1})$ for every $i \geq 0$ and every $\upsilon'_i \in \Upsilon^n$; analogously, we define a witness for the existential trace quantifiers $\exists\pi_1\exists\pi_2 \dots \exists\pi_m$ as the sequence of inputs $\upsilon = \upsilon_0\upsilon_1 \dots \in (\Upsilon^m)^\omega$ such that $\upsilon_i = \sigma(\upsilon'_0\upsilon'_1 \dots \upsilon'_{i-1})$ for every $i \geq 0$ and every $\upsilon'_i \in \Upsilon^0$. □

An application of the theorem reduces the verification problem of a HyperLTL formula with one quantifier alternation to the verification problem of a universal HyperLTL formula. If a sufficiently small strategy can be found, the reduction in complexity is substantial:

**Corollary 1 (Model checking with Given Strategies).** *The model checking problem for HyperLTL formulas with one quantifier alternation and given strategies for the existential quantifiers is in PSPACE in the size of the formula and NLOGSPACE in the size of the product of the strategy and the system.*

Note that the converse of Theorem 4 is not in general true. The satisfaction of a $\forall^*\exists^*$ HyperLTL formula does not imply the existence of a strategy, because at any given point in time the strategy only knows about a finite prefix of the universally quantified traces. Consider the formula $\forall\pi\exists\pi'.\bigcirc a_\pi \leftrightarrow a_{\pi'}$ and a system that can produce arbitrary sequences of $a$ and $\neg a$. Although the system satisfies the formula, it is not possible to give a strategy that allows us to prove this fact. Whatever choice our strategy makes, the next move of the $\forall$-player can make sure that the strategy's choice was wrong. In the following, we present a method that addresses this problem.

**Prophecy Variables.** A classic technique for resolving future dependencies is the introduction of *prophecy variables* [1]. Prophecy variables are auxiliary variables that are added to the system without affecting the behavior of the system. Such variables can be used to make predictions about the future.

We use prophecy variables to define strategies that depend on the future. In the example discussed above, $\forall\pi\exists\pi'.\bigcirc a_\pi \leftrightarrow a_{\pi'}$, the choice of the value of $a_{\pi'}$ in

the first position depends on the value of $a_\pi$ in the second position. We introduce a prophecy variable $p$ that predicts in the first position whether $a_\pi$ is true in the second position. With the prophecy variable, there exists a strategy that correctly assigns the value of $p$ whenever the prediction is correct: The strategy chooses to set $a_{\pi'}$ if, and only if, $p$ holds.

Technically, the proof technique introduces a set of fresh input variables $P$ into the system. For a $\Gamma$-labeled $\Upsilon$-transition system $\mathcal{S} = \langle S, s_0, \tau, l\rangle$, we define the $\Gamma$-labeled $(\Upsilon \cup P)$-transition system $\mathcal{S}^P = \langle S, s_0, \tau^P, l\rangle$ including the inputs $P$ where $\tau^P \colon S \times (\Upsilon \cup P) \to S$. For all $s \in S$ and $v^P \in \Upsilon \cup P$, $\tau^P(s, v^P) = \tau(s, v)$ for $v \in \Upsilon$ obtained by removing the variables in $P$ from $v^P$ (i.e., $v =_{\setminus P} v^P$). Moreover, the proof technique modifies the specification so that the original property only needs to be satisfied if the prediction is actually correct. We obtain the modified specification $\forall\pi\exists\pi'.(p_\pi \leftrightarrow \bigcirc a_\pi) \rightarrow (\bigcirc a_\pi \leftrightarrow a_{\pi'})$ in our example. The following theorem describes the general technique for one prophecy variable.

**Theorem 5 (Model checking with Prophecy Variables).** *For a transition system $\mathcal{S}$ and a quantifier-free formula $\varphi$, let $\psi$ be a quantifier-free formula over the universally quantified trace variables $\pi_1, \pi_2 \dots \pi_n$ and let $p$ be a fresh atomic proposition. It holds that $\mathcal{S} \vDash \forall\pi_1\forall\pi_2 \dots \forall\pi_n.\ \exists\pi'_1\exists\pi'_2 \dots \exists\pi'_m.\ \varphi$ if, and only if, $\mathcal{S}^{\{p\}} \vDash \forall\pi_1\forall\pi_2 \dots \forall\pi_n.\ \exists\pi'_1\exists\pi'_2 \dots \exists\pi'_m.\ \square(p_{\pi_1} \leftrightarrow \psi) \rightarrow \varphi$.*

Note that $\psi$ is restricted to refer only to *universally* quantified trace variables. Without this restriction, the method would not be sound. In our example, $\psi = a_{\pi'}$ would lead to the modified formula $\forall\pi\exists\pi'.(p_\pi \leftrightarrow a_{\pi'}) \rightarrow (\bigcirc a_\pi \leftrightarrow a_{\pi'})$, which could be satisfied with the strategy that assigns $a_{\pi'}$ to *true* iff $p_\pi$ is *false*, and thus falsifies the assumption that the prediction is correct, rather than ensuring that the original formula is true.

*Proof.* It is easy to see that the original specification implies the modified specification, since the original formula is the conclusion of the implication. Assume that the modified specification holds. Since the prophecy variable $p$ is a fresh atomic proposition, and $\psi$ does not refer to the existentially chosen traces, we can, for every choice of the universally quantified traces, always choose the value of $p$ such that it guesses correctly, i.e., that $p$ is true whenever $\psi$ holds. In this case, the conclusion and therefore the original specification must be true. □

Unfortunately, prophecy variables do not provide a complete proof technique. Consider a system allowing arbitrary sequences of $a$ and $b$ and this specification:

$$\begin{aligned}\forall\pi\exists\pi'.b_{\pi'} \wedge \square(b_{\pi'} \leftrightarrow \bigcirc\neg b_{\pi'})\\ \wedge (a_{\pi'} \rightarrow (a_\pi\ \mathcal{W}\ (b_{\pi'} \wedge \neg a_\pi)))\\ \wedge (\neg a_{\pi'} \rightarrow (a_\pi\ \mathcal{W}\ (\neg b_{\pi'} \wedge \neg a_\pi)))\end{aligned}$$

Intuitively, $\pi'$ has to be able to predict whether $\pi$ will stop outputting $a$ at an even or odd position of the trace. There is no HyperLTL formula to be used as $\psi$ in Theorem 5, because, like LTL, HyperLTL can only express non-counting properties. It is worth noting that in our practical experiments, the

incompleteness was never a problem. In many cases, it is not even necessary to add prophecy variables at all. The presented proof technique is, thus, practically useful despite this incompleteness result.

### 3.2 Model Checking with Synthesized Strategies

We now extend the model checking approach with the automatic synthesis of the strategies for the existential quantifiers. For a given HyperLTL formula of the form $\forall^n \exists^m \varphi$ and a transition system $\mathcal{S}$, we search for a transition system $\mathcal{S}_\exists = \langle X, x_0, \mu, l_\exists \rangle$, where $X$ is a set of states, $x_0 \in X$ is the designated initial state, $\mu \colon X \times \Upsilon^n \to X$ is the transition function, and $l_\exists \colon X \to \Upsilon^m$ is the labeling function, such that $\mathcal{S}^n \times (\mathcal{S}^m \mid\mid \mathcal{S}_\exists) \vDash \mathit{zip}(\varphi)$. (Since for formulas of the form $\exists^m \forall^n \varphi$ the problem only differs in the input of $\mathcal{S}_\exists$, we focus on $\forall\exists$ HyperLTL.)

**Theorem 6.** *The strategy realizability problem for $\forall^* \exists^*$ formulas is* 2ExpTime*-complete.*

*Proof (Sketch).* We reduce the strategy synthesis problem to the problem of synthesizing a distributed reactive system with a single black-box process. This problem is decidable [19] and can be solved in 2ExpTime. The lower bound follows from the LTL realizability problem [30]. □

The decidability result implies that there is an upper bound on the size of $\mathcal{S}_\exists$ that is doubly exponential in $\varphi$. Thus, the bounded synthesis approach [20] can be used to search for increasingly larger implementations, until a solution is found or the maximal bound is reached, yielding an efficient decision procedure for the strategy synthesis problem. In the following, we describe this approach in detail.

**Bounded Synthesis of Strategies.** We transform the synthesis problem into an SMT constraint satisfaction problem, where we leave the representation of strategies uninterpreted and challenge the solver to provide an interpretation. Given a HyperLTL formula $\forall^n \exists^m \varphi$ where $\varphi$ is quantifier-free, the model checking is based on the product of the $n$-fold self composition of the transition system $\mathcal{S}$, the $m$-fold self-composition of $\mathcal{S}$ where the strategy $\mathcal{S}_\exists$ controls the inputs, and the universal co-Büchi automaton $\mathcal{A}_\varphi$ representing the language $\mathcal{L}(\varphi)$ of $\varphi$.

For a quantifier-free HyperLTL formula $\varphi$, we construct the universal co-Büchi automaton $\mathcal{A}_\varphi$ such that $\mathcal{L}(\mathcal{A}_\varphi)$ is the set of words $w$ such that $\mathit{unzip}(w) \vDash \varphi$, i.e., the tuple of traces satisfies $\varphi$. We get this automaton by dualizing the non-deterministic Büchi automaton for $\neg\psi$ [6], i.e., changing the branching from non-deterministic to universal and the acceptance condition from Büchi to co-Büchi. Hence, $\mathcal{S}$ satisfies a universal HyperLTL formula $\forall \pi_1 \ldots \forall \pi_n . \varphi$ if the traces generated by the self-composition $\mathcal{S}^n$ are a subset of $\mathcal{L}(\mathcal{A}_\varphi)$.

In more detail, the algorithm searches for a transition system $\mathcal{S}_\exists = \langle X, x_0, \mu, l_\exists \rangle$ such that the run graph of $\mathcal{S}^n$, $\mathcal{S}^m \mid\mid \mathcal{S}_\exists$, and $\mathcal{A}_\varphi$, written $\mathcal{S}^n \times (\mathcal{S}^m \mid\mid \mathcal{S}_\exists) \times \mathcal{A}_\varphi$, is accepting. Formally, given a $\Gamma$-labeled $\Upsilon$-transition

system $\mathcal{S} = \langle S, s_0, \tau, l\rangle$ and a universal co-Büchi automaton $\mathcal{A}_\varphi = \langle Q, q_0, \delta, F\rangle$, where $\delta\colon Q \times \Upsilon^{n+m} \times \Gamma^{n+m} \to 2^Q$, the run graph $\mathcal{S}^n \times (\mathcal{S}^m \mid\mid \mathcal{S}_\exists) \times \mathcal{A}_\varphi$ is the directed graph $(V, E)$, with the set of vertices $V = S^n \times S^m \times X \times Q$, initial vertex $v_{init} = ((s_0, \dots, s_0), (s_0, \dots, s_0), x_0, q_0)$ and the edge relation $E \subseteq V \times V$ satisfying $((\boldsymbol{s_n}, \boldsymbol{s_m}, x, q), (\boldsymbol{s'_n}, \boldsymbol{s'_m}, x', q')) \in E$ if, and only if

$$\exists \boldsymbol{\upsilon} \in \Upsilon^n.\ \left(\boldsymbol{s_n} \xrightarrow[\tau_n]{\boldsymbol{\upsilon}} \boldsymbol{s'_n}\right) \wedge \left(\boldsymbol{s_m} \xrightarrow[\tau_m]{l_\exists(x)} \boldsymbol{s'_m}\right) \wedge \left(x \xrightarrow[\mu]{\boldsymbol{\upsilon}} x'\right)$$
$$\wedge\, q' \in \delta(q, \boldsymbol{\upsilon} \cdot l_\exists(x), l_n(\boldsymbol{s_n}) \cdot l_m(\boldsymbol{s_m})).$$

**Theorem 7.** *Given $\mathcal{S}$, $\mathcal{S}_\exists$, and a HyperLTL formula $\forall^n \exists^m \varphi$ where $\varphi$ is quantifier-free. Let $\mathcal{A}_\varphi$ be the universal co-Büchi automaton for $\varphi$. If the run graph $\mathcal{S}^n \times (\mathcal{S}^m \mid\mid \mathcal{S}_\exists) \times \mathcal{A}_\varphi$ is accepting, then $\mathcal{S} \vDash \forall^n \exists^m \varphi$.*

*Proof.* Follows from Theorem 4 and the fact that $\mathcal{A}_\varphi$ represents $\mathcal{L}(\varphi)$. □

The acceptance of a run graph is witnessed by an annotation $\lambda\colon V \to \mathbb{N} \cup \{\bot\}$ which is a function mapping every reachable vertex $v \in V$ in the run graph to a natural number $\lambda(v)$, i.e., $\lambda(v) \neq \bot$. Intuitively, $\lambda(v)$ returns the number of visits to rejecting states on any path from the initial vertex $v_{init}$ to $v$. If we can bound this number for every reachable vertex, the annotation is *valid* and the run graph is accepting. Formally, an annotation $\lambda$ is valid, if (1) the initial state is reachable ($\lambda(v_{init}) \neq \bot$) and (2) for every $(v, v') \in E$ with $\lambda(v) \neq \bot$ it holds that $\lambda(v') \neq \bot$ and $\lambda(v) \rhd \lambda(v')$ where $\trianglerighteq$ is $>$ if $v'$ is rejecting and $\geq$ otherwise. Such an annotation exists if, and only if, the run graph is accepting [20].

We encode the search for $\mathcal{S}_\exists$ and the annotation $\lambda$ as an SMT constraint system. Therefore, we use uninterpreted function symbols to encode $\mathcal{S}_\exists$ and $\lambda$. A transition system $\mathcal{S}$ is represented in the constraint system by two functions, the transition function $\tau\colon S \times \Upsilon \to S$ and the labeling function $l\colon S \to \Gamma$. The annotation is split into two parts, a reachability constraint $\lambda^{\mathbb{B}}\colon V \to \mathbb{B}$ indicating whether a state in the run graph is reachable and a counter $\lambda^{\#}\colon V \to \mathbb{N}$ that maps every reachable vertex $v$ to the maximal number of rejecting states $\lambda^{\#}(v)$ visited by any path from the initial vertex to $v$. The resulting constraint asserts that there is a transition system $\mathcal{S}_\exists$ with an accepting run graph. Note, that the functions representing the system $\mathcal{S}$ ($\tau\colon S \times \Upsilon \to S$ and $l\colon S \to \Gamma$) are given, that is, they are interpreted.

$$\begin{aligned}
&\exists \lambda^{\mathbb{B}}\colon S^n \times S^m \times X \times Q \to \mathbb{B}.\ \exists \lambda^{\mathbb{N}}\colon S^n \times S^m \times X \times Q \to \mathbb{N}.\\
&\exists \mu\colon X \times \Upsilon^n \to X.\ \exists l_\exists\colon X \to \Upsilon^m\\
&\forall \boldsymbol{\upsilon} \in \Upsilon^n.\ \forall \boldsymbol{s_n}, \boldsymbol{s'_n} \in S^n.\ \forall \boldsymbol{s_m}, \boldsymbol{s'_m} \in S^m.\ \forall q, q' \in Q.\ \forall x, x' \in X.\\
&\lambda^{\mathbb{B}}((s_0, \dots, s_0), (s_0, \dots, s_0), x_0, q_0) \wedge\\
&\Big(\lambda^{\mathbb{B}}(\boldsymbol{s_n}, \boldsymbol{s_m}, x, q) \wedge q' \in \delta(q, (\boldsymbol{\upsilon} \cdot l_\exists(x)), (l \circ (\boldsymbol{s_n} \cdot \boldsymbol{s_m}))) \wedge x' = \mu(x, \boldsymbol{\upsilon})\\
&\quad \wedge \boldsymbol{s'_n} = \tau_n(\boldsymbol{s_n}, \boldsymbol{\upsilon}) \wedge \boldsymbol{s'_m} = \tau_m(\boldsymbol{s_m}, l_\exists(x))\Big)\\
&\Rightarrow \lambda^{\mathbb{B}}(\boldsymbol{s'_n}, \boldsymbol{s'_m}, x', q') \wedge \lambda^{\mathbb{N}}(\boldsymbol{s_n}, \boldsymbol{s_m}, x, q) \trianglerighteq \lambda^{\mathbb{N}}(\boldsymbol{s'_n}, \boldsymbol{s'_m}, x', q')
\end{aligned}$$

where $\trianglerighteq$ is $>$ if $q' \in F$ and $\geq$ otherwise. The *bounded synthesis algorithm* increases the bound of the strategy $\mathcal{S}_\exists$ until either the constraints system becomes satisfiable, or a given upper bound is reached. In the case the constraint system is satisfiable, we can extract interpretations for the functions $\mu$ and $l_\exists$ using a solver that is able to produce models. These functions then represent the synthesized transition system $\mathcal{S}_\exists$.

**Corollary 2.** *Given $\mathcal{S}$ and a HyperLTL formula $\forall^*\exists^*\varphi$ where $\varphi$ is quantifier-free. If the constraint system is satisfiable for some bound on the size of $\mathcal{S}_\exists$ then $\mathcal{S} \vDash \forall^*\exists^*\varphi$.*

*Proof.* Follows immediately by Theorem 7. □

As the decision problem is decidable, we know that there is an upper bound on the size of a realizing $\mathcal{S}_\exists$ and, thus, the bounded synthesis approach is a decision procedure for the strategy realizability problem.

**Corollary 3.** *The bounded synthesis algorithm decides the strategy realizability problem for $\forall^*\exists^*$ HyperLTL.*

*Proof.* The existence of such an upper bound follows from Theorem 6. □

**Approximating Prophecy.** We introduce a new parameter to the strategy synthesis problem to approximate the information about the future that can be captured using prophecy variables. This bound represents a constant *lookahead* into future choices made by the environment. In other words, for a given $k \geq 0$, the strategy $\mathcal{S}_\exists$ is allowed to depend on choices of the $\forall$-player in the next $k$ steps. While constant lookahead is only an approximation of infinite clairvoyance, it suffices for many practical situations as shown by prior case studies [9,18].

We present a solution to synthesizing transition systems with constant lookahead for $k \geq 0$ using bounded synthesis. To simplify the presentation, we present the stand-alone problem with respect to a specification given as a universal co-Büchi automaton. The integration into the constraint system for the $\forall^*\exists^*$ HyperLTL synthesis as presented in the previous section is then straightforward. First, we present an extension to the transition system model that incorporates the notion of constant lookahead. The idea of this extension is to replace the initial state $s_0$ by a function $\mathit{init}\colon \Upsilon^k \to S$ that maps input sequences of length $k$ to some state. Thus, the transition system observes the first $k$ inputs, chooses some initial state based on those inputs, and then progresses with the same pace as the input sequence. Next, we define the run graph of such a system $\mathcal{S}_k = \langle S, \mathit{init}, \tau, l\rangle$ and an automaton $\mathcal{A} = \langle Q, q_0, \delta, F\rangle$, where $\delta\colon Q \times \Upsilon \times \Gamma \to Q$, as the directed graph $(V, E)$ with the set of vertices $V = S \times Q \times \Upsilon^k$, the initial vertices $(s, q_0, \boldsymbol{v}) \in V$ such that $s = \mathit{init}(\boldsymbol{v})$ for every $\boldsymbol{v} \in \Upsilon^k$, and the edge relation $E \subseteq V \times V$ satisfying $((s, q, v_1v_2\cdots v_k), (s', q', v_1'v_2'\cdots v_k')) \in E$ if, and only if

$$\exists v_{k+1} \in \Upsilon.\, s \xrightarrow{v_{k+1}} s' \wedge q' \in \delta(q, v_1, l(s)) \wedge \bigwedge_{1 \leq i \leq k} v_i' = v_{i+1}.$$

**Lemma 1.** *Given a universal co-Büchi automaton $\mathcal{A}$ and a $k$-lookahead transition system $\mathcal{S}_k$. $\mathcal{S}_k \vDash \mathcal{A}$ if, and only if, the run graph $\mathcal{S}_k \times \mathcal{A}$ is accepting.*

Finally, synthesis amounts to solving the following constraint system:

$$
\begin{aligned}
&\exists \lambda^{\mathbb{B}} \colon S \times Q \times \Upsilon^k \to \mathbb{B}.\, \exists \lambda^{\mathbb{N}} \colon S \times Q \times \Upsilon^k \to \mathbb{N}. \\
&\exists \mathit{init} \colon \Upsilon^k \to S.\, \exists \tau \colon S \times \Upsilon \to S.\, \exists l \colon S \to \Gamma. \\
&(\forall \boldsymbol{\upsilon} \in \Upsilon^k.\, \lambda^{\mathbb{B}}(\mathit{init}(\boldsymbol{\upsilon}), q_0, \boldsymbol{\upsilon})) \wedge \\
&\forall \upsilon_1 \upsilon_2 \cdots \upsilon_{k+1} \in \Upsilon^{k+1}.\, \forall s, s' \in S.\, \forall q, q' \in Q. \\
&\big(\lambda^{\mathbb{B}}(s, q, \upsilon_1 \cdots \upsilon_k) \wedge s' = \tau(s, \upsilon_{k+1}) \wedge q' \in \delta(q, \upsilon_1, l(s))\big) \\
&\Rightarrow \lambda^{\mathbb{B}}(s', q', \upsilon_2 \cdots \upsilon_{k+1}) \wedge \lambda^{\mathbb{N}}(s, q, \upsilon_1 \cdots \upsilon_k) \trianglerighteq \lambda^{\mathbb{N}}(s', q', \upsilon_2 \cdots \upsilon_{k+1})
\end{aligned}
$$

**Corollary 4.** *Given some $k \geq 0$, if the constraint system is satisfiable for some bound on the size of $\mathcal{S}_k$ then $\mathcal{S}_k \vDash \mathcal{A}$.*

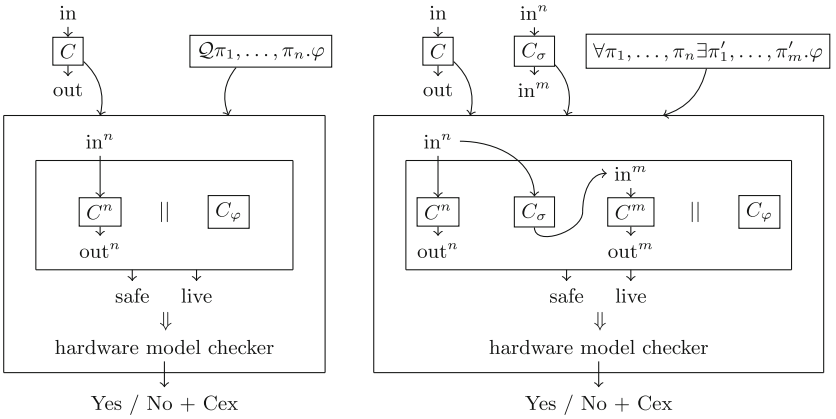
## 4 Synthesis with Quantifier Alternations

We now build on the introduced techniques to solve the *synthesis* problem for HyperLTL with quantifier alternation, that is, we search for implementations that satisfy the given properties. In previous work [13], the synthesis problem for $\exists^*\forall^*$ HyperLTL was solved by a reduction to the distributed synthesis problem. We present an alternative synthesis procedure that (1) introduces the necessary concepts for the synthesis of the $\forall^*\exists^*$ fragment and that (2) strictly decomposes the choice of the existential trace quantifier from the implementation.

Fix a formula of the form $\exists^m \forall^n \varphi$. We again reduce the verification problem to the problem of determining whether a run graph is accepting. As the existential quantifiers do not depend on the universal ones, there is no future dependency and thus no need for prophecy variables or bounded lookahead. Formally, $\mathcal{S}_\exists$ is a tuple $\langle X, x_0, \mu, l_\exists \rangle$ such that $X$ is a set of states, $x_0 \in X$ is the designated initial state, $\mu \colon X \to X$ is the transition function, and $l_\exists \colon X \to \Upsilon^m$ is the labeling function. $\mathcal{S}_\exists$ produces infinite sequences of $(\Upsilon^m)^\omega$, without having any knowledge about the behavior of the universally quantified traces. The run graph is then $(\mathcal{S}^m \mid\mid \mathcal{S}_\exists) \times \mathcal{S}^n \times \mathcal{A}_\varphi$. The constraint system is built analogously to Sect. 3.2, with the difference that the representation of the system $\mathcal{S}$ is now also uninterpreted. In the resulting SMT constraint system, we have two bounds, one for the size of the implementation $\mathcal{S}$ and one for the size of $\mathcal{S}_\exists$.

**Corollary 5.** *The bounded synthesis algorithm decides the realizability problem for $\exists^*\forall^1$ HyperLTL and is a semi-decision procedure for $\exists^*\forall^{>1}$ HyperLTL.*

The synthesis problem for formulas in the $\forall^*\exists^*$ HyperLTL fragment uses the same reduction to a constraint system as the strategy synthesis in Sect. 3.2, with the only difference that the transition system $\mathcal{S}$ itself is uninterpreted. In the resulting SMT constraint systems, we have three bounds, the size of the implementation $\mathcal{S}$, the size of the strategy $\mathcal{S}_\exists$, and the lookahead $k$.




(a) Alternation-free model checking with self-composition

(b) One-alternation model checking with given strategy (case $\forall^n\exists^m$ HyperLTL)

**Fig. 1.** HyperLTL model checking with MCHyper

**Corollary 6.** *Given a HyperLTL formula $\forall^n\exists^m\varphi$ where $\varphi$ is quantifier-free. $\forall^n\exists^m\varphi$ is realizable if the SMT constraint system corresponding to the run graph $\mathcal{S}^n \times (\mathcal{S}^m \mid\mid \mathcal{S}_\exists) \times \mathcal{A}_\varphi$ is satisfiable for some bounds on $\mathcal{S}$, $\mathcal{S}_\exists$, and lookahead $k$.*

## 5 Implementations and Experimental Evaluation

We have integrated the model checking technique with a manually provided strategy into the HyperLTL hardware model checker MCHyper[1]. For the synthesis of strategies and reactive systems from hyperproperties, we have developed a separate bounded synthesis tool based on SMT-solving. In the following, we describe these implementations and report on experimental results. All experiments ran on a machine with dual-core Core i7, 3.3 GHz, and 16 GB memory.

**Hardware Model Checking with Given Strategies.** We have extended the model checker MCHyper [18] from the alternation-free fragment to formulas with one quantifier alternation. The input to MCHyper is a circuit description as an And-Inverter-Graph in the Aiger format and a HyperLTL formula. Figures 1a and 1 show the model checking process in MCHyper without and with quantifier alternation, respectively. For formulas with quantifier alternation, the model checker now also accepts a strategy as an additional Aiger circuit $C_\sigma$. Based on this strategy, MCHyper creates a new circuit where only the inputs of the universal system copies are exposed and the inputs of the existential system

[1] Try the online tool interface with the latest version of MCHyper: https://www.react.uni-saarland.de/tools/online/MCHyper/.

**Table 1.** Experimental results for MCHyper on the software doping and mutual exclusion benchmarks. All experiments used the IC3 option for abc. Model and property names correspond to the ones used in [9] and [18].

| Model | #Latches | Property | Time[s] |
|---|---|---|---|
| EC 0.05 | 17 | $(10.a) + (10.b)$ | 1.8 |
| EC 0.00625 | 23 | $(10.a) + (10.b)$ | 53.4 |
| AEC 0.05 | 19 | $(\neg 10.a) + (\neg 10.b)$ | 2.8 |
| AEC 0.00625 | 25 | $(\neg 10.a) + (\neg 10.b)$ | 160.1 |
| Bakery.a.n.s | 47 | Sym5 | 50.6 |
| | | Sym6 | 27.5 |
| Bakery.a.n.s.5proc | 90 | Sym7 | 461.3 |
| | | Sym8 | 472.3 |

copies are determined by the strategy. The new circuit is then model checked as described in [18] with abc [4].

We evaluate our extension of MCHyper on formulas with quantifier alternation based on benchmarks from software doping [9] and symmetry in mutual exclusion algorithms [18]. Both considered problems have previously been analyzed with MCHyper; however, since the properties in both problems require quantifier alternation, we were previously limited to a (manually obtained) approximation of the properties as universal formulas. The correctness of manual approximations is not given but has to be shown separately. By directly model checking the formula with quantifier alternation we know that we are checking the correct formula without needing any additional proof of correctness.

*Software Doping.* D'Argenio et al. [9] examined a clean and a doped version of an emission control program of a car and used the previous version of MCHyper to formally verify approximations of these properties. Robust cleanness is expressed in the one-alternation fragment using two $\forall^2\exists^1$ HyperLTL formulas (given in Prop. 19 in [9], cf. Sect. 1). In [9], the formulas were strengthened into alternation-free formulas that imply the original properties. Despite the quantifier alternation, Table 1 shows that the new version of MCHyper verifies the precise formulas in roughly the same time as the alternation-free approximations [9] while giving stronger correctness guarantees.

*Symmetry in Mutual Exclusion Protocols.* $\forall^*\exists^*$ HyperLTL allows us to specify symmetry for mutual exclusion protocols. In such protocols, we wish to guarantee that every request is eventually answered, and the grants are mutually exclusive. In our experiments, we used an implementation of the Bakery protocol [25]. Table 1 shows the verification results for the precise $\forall^1\exists^1$ properties. Comparing these results to the performance on the approximations of the symmetry properties [18], we, again, observe that the verification times are similar. However, we gain the additional correctness guarantees as described above.

**Strategy and System Synthesis.** For the synthesis of strategies for existential quantifiers and for the synthesis of reactive systems from hyperproperties, we have developed a separate bounded synthesis tool based on SMT-solving with Z3 [29]. Our evaluation is based on two benchmark families, the *dining cryptographers* problem [5] and a simplified version of the symmetry problem in mutual exclusion protocols discussed previously. The results are shown in Table 2. Obviously, synthesis operates at a vastly smaller scale than model checking with given strategies. In the dining cryptographers example, Z3 was unable to find an implementation for the full synthesis problem, but could easily synthesize strategies for the existential trace quantifiers when provided with an implementation. With the progress of constraint solver that employ quantification over Boolean functions [31] we expect scalability improvements of our synthesis approach.

**Table 2.** Summary of the experimental results on the benchmarks sets described in Sect. 5. When no hyperproperty is given, only the LTL part is used.

| Instance | Hyperproperty | $\lvert\mathcal{S}\rvert$ | $\lvert\mathcal{S}_\exists\rvert$ | Time [s] |
|---|---|---|---|---|
| Dining cryptographers | distributed + deniability | | | TO |
| | distributed + deniability with given $\mathcal{S}$ | (1) | 1 | 1.2 |
| Mutex | — | 2 | – | <1 |
| | symmetry | 3 | 1 | 3.4 |
| Mutex w/o spurious grants | — | 3 | – | <1 |
| | symmetry | 3 | 1 | 3.9 |
| | wait-free | 3 | 3 | 46 |
| | symmetry + wait-free | 3 | 1 + 3 | 840 |

## 6 Conclusions

We have presented model checking and synthesis techniques for hyperliveness properties expressed as HyperLTL formulas with quantifier alternation. The alternation makes it possible to specify hyperproperties such as generalized noninterference, symmetry, and deniability. Our approach is the first method for the synthesis of reactive systems from HyperLTL formulas with quantifier alternation and the first practical method for the verification of such specifications.

The approach is based on a game-theoretic view of existential quantifiers, where the $\exists$-player reacts to decisions of the $\forall$-player. The key advantage is that the complementation of the system automaton is avoided (cf. [18]). Instead, a strategy must be found for the $\exists$-player. Since this can be done either manually or through automatic synthesis, the user of the model checking or synthesis tool has the opportunity to trade some automation for a significant gain in performance.

**Acknowledgements.** We would like to thank Sebastian Biewer for providing the software doping models and formulas, Marvin Stenger for his advice on our synthesis experiments, and Jana Hofmann for her helpful comments on a draft of this paper.

## References


1. Abadi, M., Lamport, L.: The existence of refinement mappings. Theor. Comput. Sci. **82**(2), 253–284 (1991). https://doi.org/10.1016/0304-3975(91)90224-P
2. Barthe, G., Crespo, J.M., Kunz, C.: Beyond 2-safety: asymmetric product programs for relational program verification. In: Artemov, S., Nerode, A. (eds.) LFCS 2013. LNCS, vol. 7734, pp. 29–43. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35722-0_3
3. Barthe, G., D'Argenio, P.R., Rezk, T.: Secure information flow by self-composition. In: Proceedings of CSFW, pp. 100–114. IEEE Computer Society (2004). https://doi.org/10.1109/CSFW.2004.17
4. Brayton, R., Mishchenko, A.: ABC: an academic industrial-strength verification tool. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 24–40. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_5
5. Chaum, D.: Security without identification: transaction systems to make big brother obsolete. Commun. ACM **28**(10), 1030–1044 (1985). https://doi.org/10.1145/4372.4373
6. Clarkson, M.R., Finkbeiner, B., Koleini, M., Micinski, K.K., Rabe, M.N., Sánchez, C.: Temporal logics for hyperproperties. In: Abadi, M., Kremer, S. (eds.) POST 2014. LNCS, vol. 8414, pp. 265–284. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54792-8_15
7. Clarkson, M.R., Schneider, F.B.: Hyperproperties. J. Comput. Secur. **18**(6), 1157–1210 (2010). https://doi.org/10.3233/JCS-2009-0393
8. Cook, B., Khlaaf, H., Piterman, N.: On automation of CTL* verification for infinite-state systems. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 13–29. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_2
9. D'Argenio, P.R., Barthe, G., Biewer, S., Finkbeiner, B., Hermanns, H.: Is your software on dope? - formal analysis of surreptitiously "enhanced" programs. In: Yang, H. (ed.) ESOP 2017. LNCS, vol. 10201, pp. 83–110. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54434-1_4
10. D'Souza, D., Holla, R., Raghavendra, K.R., Sprick, B.: Model-checking trace-based information flow properties. J. Comput. Secur. **19**(1), 101–138 (2011). https://doi.org/10.3233/JCS-2010-0400
11. Finkbeiner, B., Hahn, C.: Deciding hyperproperties. In: Proceedings of CONCUR. LIPIcs, vol. 59, pp. 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016). https://doi.org/10.4230/LIPIcs.CONCUR.2016.13
12. Finkbeiner, B., Hahn, C., Hans, T.: MGHyper: checking satisfiability of HyperLTL formulas beyond the $\exists^*\forall^*$ fragment. In: Lahiri, S.K., Wang, C. (eds.) ATVA 2018. LNCS, vol. 11138, pp. 521–527. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4_31
13. Finkbeiner, B., Hahn, C., Lukert, P., Stenger, M., Tentrup, L.: Synthesizing reactive systems from hyperproperties. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 289–306. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_16

14. Finkbeiner, B., Hahn, C., Stenger, M.: EAHyper: satisfiability, implication, and equivalence checking of hyperproperties. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 564–570. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_29
15. Finkbeiner, B., Hahn, C., Stenger, M., Tentrup, L.: Monitoring hyperproperties. In: Lahiri, S., Reger, G. (eds.) RV 2017. LNCS, vol. 10548, pp. 190–207. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67531-2_12
16. Finkbeiner, B., Hahn, C., Stenger, M., Tentrup, L.: RVHyper: a runtime verification tool for temporal hyperproperties. In: Beyer, D., Huisman, M. (eds.) TACAS 2018. LNCS, vol. 10806, pp. 194–200. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89963-3_11
17. Finkbeiner, B., Hahn, C., Torfah, H.: Model checking quantitative hyperproperties. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 144–163. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_8
18. Finkbeiner, B., Rabe, M.N., Sánchez, C.: Algorithms for model checking HyperLTL and HyperCTL*. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 30–48. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_3
19. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: Proceedings of LICS, pp. 321–330. IEEE Computer Society (2005). https://doi.org/10.1109/LICS.2005.53
20. Finkbeiner, B., Schewe, S.: Bounded synthesis. STTT **15**(5–6), 519–539 (2013). https://doi.org/10.1007/s10009-012-0228-z
21. Goguen, J.A., Meseguer, J.: Security policies and security models. In: Proceedings of S&P, pp. 11–20. IEEE Computer Society (1982). https://doi.org/10.1109/SP.1982.10014
22. Hahn, C., Stenger, M., Tentrup, L.: Constraint-based monitoring of hyperproperties. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11428, pp. 115–131. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17465-1_7
23. Huisman, M., Worah, P., Sunesen, K.: A temporal logic characterisation of observational determinism. In: Proceedings of CSFW, p. 3. IEEE Computer Society (2006). https://doi.org/10.1109/CSFW.2006.6
24. Klein, F., Zimmermann, M.: How much lookahead is needed to win infinite games? In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 452–463. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47666-6_36
25. Lamport, L.: A new solution of Dijkstra's concurrent programming problem. Commun. ACM **17**(8), 453–455 (1974). https://doi.org/10.1145/361082.361093
26. Lynch, N.A., Vaandrager, F.W.: Forward and backward simulations: I. untimed systems. Inf. Comput. **121**(2), 214–233 (1995). https://doi.org/10.1006/inco.1995.1134
27. McCullough, D.: Noninterference and the composability of security properties. In: Proceedings of S&P, pp. 177–186. IEEE Computer Society (1988). https://doi.org/10.1109/SECPRI.1988.8110
28. van der Meyden, R., Zhang, C.: Algorithmic verification of noninterference properties. Electr. Notes Theor. Comput. Sci. **168**, 61–75 (2007). https://doi.org/10.1016/j.entcs.2006.11.002

29. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
30. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proceedings of POPL, pp. 179–190. ACM Press (1989). https://doi.org/10.1145/75277.75293
31. Tentrup, L., Rabe, M.N.: Clausal abstraction for DQBF. In: Janota, M., Lynce, I. (eds.) SAT 2019. LNCS, vol. 11628, pp. 388–405. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24258-9_27