2022 IEEE 35th Computer Security Foundations Symposium (CSF)

2022 IEEE 35th Computer Security Foundations Symposium (CSF) | 978-1-6654-8417-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/CSF54842.2022.9919658

Prophecy Variables for Hyperproperty Verification

Raven Beutner
CISPA Helmholtz Center for
Information Security
Germany

Bernd Finkbeiner
CISPA Helmholtz Center for
Information Security
Germany

***Abstract*—Temporal logics for hyperproperties like HyperLTL use trace quantifiers to express properties that relate multiple system runs. In practice, the verification of such specifications is mostly limited to formulas without quantifier alternation, where verification can be reduced to checking a trace property over the self-composition of the system. Quantifier alternations like $\forall\pi.\exists\pi'.\phi$, can either be solved by complementation or with an interpretation as a two-person game between a $\forall$-player, who incrementally constructs the trace π, and an $\exists$-player, who constructs π' in such a way that π and π' together satisfy ϕ. The game-based approach is significantly cheaper but incomplete because the $\exists$-player does not know the future moves of the $\forall$-player. In this paper, we establish that the game-based approach can be made complete by adding (ω-regular) temporal prophecies. Our proof is constructive, yielding an effective algorithm for the generation of a complete set of prophecies.**

***Index Terms*—Hyperproperties, HyperLTL, Hyperliveness, Verification, Prophecy Variables, Completeness**

I. Introduction

Hyperproperties [1] are system properties that relate multiple execution traces in a system and commonly arise, e.g., in information-flow policies. An increasingly popular logic for the specification of general hyperproperties is HyperLTL [2], which extends linear-time temporal logic (LTL) with explicit trace quantification. In HyperLTL we can, for example, express a simple variant of non-interference (NI) [3] as follows:

$$\forall\pi.\forall\pi'.\square\Big(\bigwedge_{a\in L_{in}} a_\pi \leftrightarrow a_{\pi'}\Big) \rightarrow \square\Big(\bigwedge_{a\in L_{out}} a_\pi \leftrightarrow a_{\pi'}\Big)$$

Here L_{in} and L_{out} are sets of atomic propositions denoting low-security inputs and outputs. Sets H_{in} and H_{out} are the high-security counterparts. The HyperLTL property states that any two traces with identical low-security inputs have identical low-security outputs, i.e., the system behaves deterministically for a low-security user. A less strict notation of non-interference, in the literature often referred to as *generalized non-interference* (GNI) [4], can be expressed as follows:

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

GNI states that for all traces π and π', there *exists* a third trace π'' that agrees with the low-security inputs and outputs of π but with the high-security inputs of π'. Phrased differently, any input-output behavior observable by a low-security user is compatible with any sequence of high-security inputs. GNI is of particular interest as it applies to non-deterministic systems where the simple variant of NI is violated when the nondeterminism influences the low-security output.

In this paper, we study the verification of HyperLTL, i.e., the question of whether a given system satisfies a given property. For HyperLTL, the structure of the quantifier prefix has direct implications on the complexity of the verification problem. For our example properties, the fundamental difference (w.r.t. verification) between NI and GNI, is that NI uses only universal quantification over traces (we say NI is alternation-free) whereas GNI involves a quantifier alternation. Verification of alternation-free properties is well understood and is reducible to the verification of a trace property on a suitable self-composition of the system [5], [6]. By contrast, verification of properties involving alternations is much more challenging. In the complementation-based approach [6] a quantifier alteration like $\forall\pi.\exists\pi'.\phi$ is interpreted as $\forall\pi.\neg\forall\pi'.\neg\phi$ which can be checked by incrementally eliminating quantifiers with interposed system complementation. This complementation is infeasible for larger systems.

A. Strategy-based Verification

A first scalable verification method for $\forall^*\exists^*$ HyperLTL properties (i.e., properties that involve an arbitrary number of universal quantifiers followed by an arbitrary number of existential quantifiers, such as GNI) has been proposed by Coenen et al. [7], which we call *strategy-based verification*. The key idea is to interpret a $\forall\pi.\exists\pi'.\phi$ formula as a game. The $\forall$-player controls the universally quantified trace by moving through the system (thereby producing a trace π) while the $\exists$-player reacts with moves in a separate copy of the system (thereby producing a trace π'). The $\exists$-player wins if π combined with π' satisfies ϕ. The resulting verification approach is sound (i.e., a winning strategy for the $\exists$-player implies that the property holds) and much cheaper than the complementation-based method (the game can be solved in polynomial time whereas the complementation incurs an exponential blow-up). The method is, however, incomplete. The $\exists$-player can, in step i, only react to the moves of the $\forall$-player up to step i (i.e., only a finite prefix of the trace constructed by the $\forall$-player) and has no access to future behavior. See Section II for examples.

B. Prophecies to the Rescue

A common proof technique to make information about future events accessible are prophecy variables [8]. In the context of hyperproperty verification, a prophecy provides the

© 2022, Raven Beutner. Under license to IEEE.
DOI 10.1109/CSF54842.2022.00030

∃-player with information about the future behavior of the ∀-player. Appropriately chosen prophecies result in the existence of a winning strategy for the ∃-player (who, in each step, has access to the prophecies), even in cases where there is no winning strategy without the prophecies [7]. However, in the context of hyperproperty verification, prophecies have, so far, been used as an *ad hoc* method where prophecies are provided by the user on a case-by-case basis [7]. With this paper, we conduct a first formal study into the expressive power of prophecies. In particular, we show that (ω-regular) prophecies are complete, i.e., prophecies always suffice for successful verification. Our main result informally reads as follows:

> *For any finite-state system $\mathcal{T}$ and $\forall^*\exists^*$ HyperLTL property φ, there exist finitely many (ω-regular) prophecies such that the ∃-player has a winning strategy (with access to the prophecies) if and only if $\mathcal{T}$ satisfies φ.*

When given such a complete set of prophecies, verification of a hyperproperty reduces (in a sound-and-complete manner) to solving a finite-state two-player game. Notably, our proof of the above result is constructive, i.e., we give an explicit (and effective) construction of a complete set of prophecies, represented as ω-automata.

C. Prototype Implementation

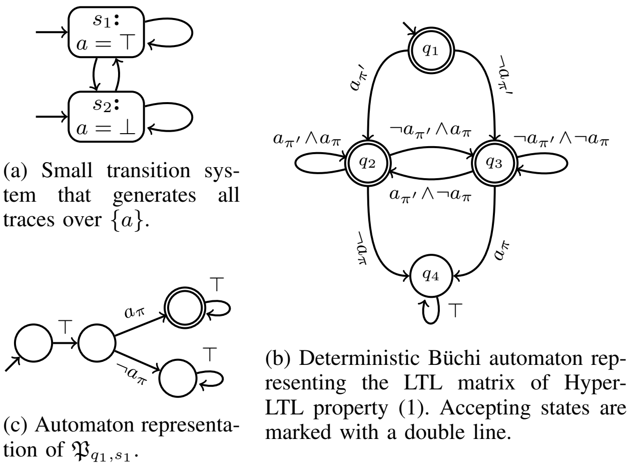
We have implemented our prophecy construction in a prototype model checker for $\forall^*\exists^*$ HyperLTL formulas, called HyPro (short for **Hy**perproperty Verification with **Pro**phecies). If required, HyPro automatically constructs a complete set of prophecies and thus constitutes the first *complete* verifier for $\forall^*\exists^*$ HyperLTL formulas with a safety matrix (see Section IX). We emphasize that this paper's main contribution is a completeness proof for prophecies in hyperproperty verification. While HyPro demonstrates that our explicit prophecy construction is applicable in practice, it is, currently, limited to small systems.

D. Structure

The remainder of this paper is structured as follows. In Section II we demonstrate the need for prophecies on a small example and outline our automatic prophecy construction. In Section III we discuss related approaches, and in Section IV define preliminaries and introduce HyperLTL. We define strategy-based verification and prophecies in Section V, and discuss completeness in Section VI. Afterward, we first outline our prophecy construction for HyperLTL specifications where the matrix is a safety property (in Section VII), and then extend it to full ω-regularity in Section VIII. In Section IX we discuss prophecy-based verification and evaluate our prototype model checker HyPro. Lastly, we outline further applications of (and future directions for) prophecy-based verification (in Section X).

II. Overview

In this section, we demonstrate the need for prophecies in hyperproperty verification on two small examples (in Sections II-A and II-B). Afterward, we sketch our automated prophecy construction (in Section II-C).



(a) Small transition system that generates all traces over $\{a\}$.

(b) Deterministic Büchi automaton representing the LTL matrix of HyperLTL property (1). Accepting states are marked with a double line.

(c) Automaton representation of $\mathfrak{P}_{q_1,s_1}$.

Fig. 1: A simple example that demonstrates why prophecies are needed for successful strategy-based verification. Figure 1c depicts the (minimized) automaton resulting from our prophecy construction.

A. Strategy-based Verification and Prophecies

As a (very) small example, consider the transition system $\mathcal{T}$ in Figure 1a, which generates all traces over atomic propositions $AP = \{a\}$, and the HyperLTL specification

$$\varphi := \forall \pi . \exists \pi' . \square (a_{\pi'} \leftrightarrow \bigcirc a_\pi). \tag{1}$$

The property states that for every trace π there should be a trace π' that mimics π *one step into the future*. Clearly $\mathcal{T} \models \varphi$, i.e., the system satisfies the property.

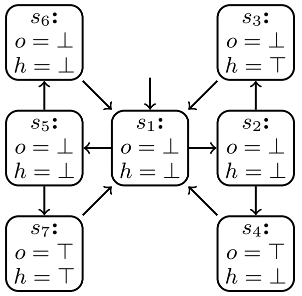
To automatically check this using strategy-based verification [7], we construct a game where, in each step, the ∀-player chooses a successor state for trace π (in the first step the ∀-player chooses any initial state), and the ∃-player reacts by choosing a successor state for trace π' (in a separate copy of the system). The ∃-player tries to construct trace π' such that π' combined with π satisfies the LTL matrix of (1). However, even though $\mathcal{T} \models \varphi$, the ∃-player loses this game. In every step of the game, the ∃-player needs to move to either s_1 or s_2. With either choice, the ∀-player can (in the next step of the game) move its copy to the opposite state (i.e., move to s_2 if the ∃-player moved to s_1 and vice versa) and thereby ensure that $a_{\pi'} \not\leftrightarrow \bigcirc a_\pi$ holds; strategy-based verification fails.

To win the game, the ∃-player would need to base its decision on the *next* move of the ∀-player. Prophecies can provide this necessary information about the future behavior of the ∀-player. Consider the LTL-definable prophecy $\xi := \bigcirc a_\pi$.[1] If the ∃-player has access to this prophecy (i.e., has access to an oracle that tells him, in each step of the game, if ξ currently holds or not), a winning strategy exists. For example, if ξ holds

[1] In our setting, a prophecy is a ω-regular set of behaviors of the universally quantified traces. If possible, we can represent this set as an LTL formula.

```
1: repeat
2:     if $\star$ then
3:         $h := \star$
4:         $o := h$
5:     else
6:         $h := \star$
7:         $o := \neg h$
```

(a) Simple example program. Here, $\star$ denotes a non-deterministic choice of a (boolean) value.


s_6: $o = \bot$ $h = \bot$
s_3: $o = \bot$ $h = \top$
s_5: $o = \bot$ $h = \bot$
s_1: $o = \bot$ $h = \bot$
s_2: $o = \bot$ $h = \bot$
s_7: $o = \top$ $h = \top$
s_4: $o = \top$ $h = \bot$

(b) A simplified transition system obtained from the program in Figure 2a.

Fig. 2: Simple example program that requires prophecies to successfully verify GNI using strategy-based verification.

(so a holds in the next step on π), the $\exists$-player moves to s_1 as this ensures $a_{\pi'} \leftrightarrow \bigcirc a_\pi$.

B. Prophecies and GNI

Prophecies are also needed when applying strategy-based verification to more realistic systems and properties. As a second example, consider the program in Figure 2a where h is a high-security input and o a low-security output, and the GNI property from Section I. Figure 2b depicts a simplified version of the program as a transition system. In state s_1 the system can non-deterministically transition into s_2 or s_5. From s_2 the values of h and o disagree (as in the second branch of the conditional in Figure 2a) whereas from s_5 the values agree (as in the first branch). It is easy to see that this program (and/or transition system) satisfies GNI, but strategy-based verification fails. In order to resolve the non-deterministic choice in line 2 of the program (or in state s_1 of the transition system), the $\exists$-player needs to know the *next* input and output on traces π and π'. Prophecies can provide the needed information about the future behavior on π, π'. This information is, for example, made available via the LTL-definable prophecies $\xi_1 := \bigcirc o_\pi$ and $\xi_2 := \bigcirc h_{\pi'}$. With access to these prophecies, a winning strategy for the $\exists$-player exists. For example, if ξ_1 holds (so the next value of o on π is $\top$) and ξ_2 does not hold (so the next value of h on π' is $\bot$), the $\exists$-player moves to s_2 as this supports a later transition to the state s_4 (where $o = \top, h = \bot$ as required). Note that, different from the example in Section II-A, this dependency on the next state is *not* explicit in the property (as GNI does not involve any $\bigcirc$s).

C. Automated Prophecy Construction

We now sketch how to *automatically* construct a complete set of prophecies, i.e., a set of prophecies that ensures that the $\exists$-player can win the game (provided the property holds).

As a concrete example, we use the system and property from Section II-A. For this example, the LTL matrix of (1) is a safety property, which simplifies the prophecy construction significantly. Conceptually, the idea is to design prophecies that directly identify those states that the $\exists$-player can move to without losing the game. We observe that the $\exists$-player can (safely) move to state s (for $s \in \{s_1, s_2\}$) iff the trace π constructed by the $\forall$-player is such that there exists *some* trace π' starting in s that serves as a witness for π. To formalize this, Figure 1b depicts a deterministic Büchi automaton $\mathcal{A}$ that tracks the matrix of (1). Given an automaton state q (for $q \in \{q_1, q_2, q_3, q_4\}$) and a system state s, we summarize all traces constructed by the $\forall$-player on which a witness trace starting in state s exists (where the automaton begins tracking in state q). Formally we define

$$\mathfrak{P}_{q,s} := \{t \in (2^{\{a\}})^\omega \mid \exists t' \in \mathit{Traces}(\mathcal{T}_s).\, t \otimes t' \in \mathcal{L}(\mathcal{A}_q)\}$$

where $\mathit{Traces}(\mathcal{T}_s)$ are all traces starting in state s, $\mathcal{L}(\mathcal{A}_q)$ are all traces accepted by $\mathcal{A}$ starting in state q, and $t \otimes t'$ is the pointwise product of t and t'. See Section VII for a formal treatment.

The resulting prophecies determine which move is safe for the $\exists$-player: If during the game the current state of the automaton tracking the matrix of (1) is q, and prophecy $\mathfrak{P}_{q,s}$ holds (i.e., the trace constructed by the $\forall$-player is contained in this set), the $\exists$-player can safely pick s as its successor.

As an example, we consider the set $\mathfrak{P}_{q_1,s_1}$. By taking the product of $\mathcal{T}$ and $\mathcal{A}$, we obtain an automaton representation of $\mathfrak{P}_{q_1,s_1}$, which, after minimization, results in the Büchi automaton depicted in Figure 1c. Coincidentally, this automaton directly corresponds to the LTL prophecy $\xi := \bigcirc a_\pi$ identified in Section II-A. As we already argued in Section II-A, the single prophecy $\mathfrak{P}_{q_1,s_1}$ thus provides sufficient information for the $\exists$-player.[2]

III. Related Work

1) Hyperproperty Verification: Recently, the automated verification of hyperproperties expressed in general logics has received significant attention. Verification of alternation-free formulas (and, in particular, k-safety) is reducible to the verification of a trace property on the self-composition of the system [5], [6]. In contrast, few attempts at the automatic verification of properties involving a quantifier alteration have been made. This is in stark contrast to the fact that many relevant properties (especially in non-deterministic systems) require alternation. Examples include information-flow policies like GNI, refinement properties, fairness, and robust cleanness. Barthe et al. [9] describe an asymmetric product of the system such that only a subset of the behavior of the second system is preserved, thereby allowing the verification of $\forall\exists$ properties. It is challenging to construct an asymmetric product and verify its correctness (i.e., show that the product preserves all behavior of the first, universally quantified, system). Unno et al. [10] describe a constraint-based approach to verify functional (opposed to temporal) $\forall\exists$ properties. In

[2] In general, our completeness result (for cases where the matrix of the HyperLTL formula is a safety property) states that the set $\{\mathfrak{P}_{q,s}\}_{q \in Q, s \in S}$ where Q is the set of automaton states and S the set of system states *always* provides sufficient information for the $\exists$-player to win (provided the property holds). For properties where the matrix does not denote a safety property, a more involved construction is necessary (see Section VIII).

their framework, both the existentially quantified traces and the scheduling of the system are encoded in an extension of constraint Horn clauses. Lamport and Schneider [11] outline a deductive approach to verify hyperproperties by reducing the verification to TLA. This is possible as existential trace quantification can be internalized into the TLA specification. Hsu et al. [12] present a bounded model checking algorithm for hyperproperties. As usual for bounded approaches, a property can only be refuted if there exists a finite set of finite paths refuting it; bounded model checking for hyperproperties is incomplete. A first practical (albeit incomplete) algorithm for the verification of temporal properties involving quantifier alternation (expressed in HyperLTL) was proposed by Coenen et al. [7] in the form of strategy-based verification, which forms the basic setting of this work. Strategy-based verification is also applicable to infinite-state systems [13].

2) Prophecy Variables: Abadi and Lamport have introduced the concept of prophecies as a proof technique in the context of refinement mappings between state machines, and have shown completeness in this setting [8]. Coenen et al. [7] use prophecies to strengthen the $\exists$-player in strategy-based verification. It is important to note that the use of temporal prophecies advocated in [7] (and studied in this paper) differs from the setting of Abadi and Lamport [8] in several key regards. In [8], a prophecy variable changes the system by adding a variable that records the future behavior of the system as a sequence of states.[3] We take a different point of view: In our setting, we do not manipulate the system but define a prophecy as a ω-regular set of behavior (expressed in temporal logic). The $\exists$-player is only provided with a single bit of information that indicates if the future behavior of $\forall$-player lies within the prophecy or not.

While Coenen et al. [7] already discuss prophecies, they consider them as an *ad hoc* feature where the user must provide prophecies on a case-by-case basis. We study prophecies in the same setting (albeit our prophecies are ω-regular and not necessarily LTL-definable as in [7]) but conduct a systematic analysis of the expressiveness of strategy-based verification when enriched with prophecies. In particular, we establish that prophecies *always* suffice to verify a property and give an explicit (and fully automatic) algorithm for the construction of a complete set of prophecies. Compared to the purely semantic construction of Abadi and Lamport [8], we work in the fixed framework of ω-regularity and represent prophecies as ω-automata.

Prophecies as a proof technique have found application in various settings. They have been used for the verification of branching-time properties [14], the construction of simulations between automata [15], to strengthen proofs in program logics [16]–[18], and to construct liveness-to-safety transformations [19]. Cook and Koskinen [20] introduce prophecies in the form of decision predicates to verify LTL properties using CTL solvers on infinite-state systems. A decision predicate can be seen as a limited form of (non-boolean) temporal prophecy that predicts the *number* of occurrences of a particular event in the future. Closely related to our setting is the work by Unno et al. [10]. They show that for the verification of functional $\forall\exists$ properties, it is sufficient to have a prophecy variable that simply predicts the final state of the universally quantified execution. In our temporal setting, the prophecy construction is necessarily more complex as it needs to provide information about the temporal behavior of the universally quantified execution, and the information communicated per prophecy is restricted to a single bit.

IV. Preliminaries

We fix a set of atomic propositions AP and define $\Sigma := 2^{AP}$. A trace is an element $t \in \Sigma^\omega$. We write $t(i)$ to denote the ith element (starting with 0) and $t[i, \infty]$ for the infinite suffix starting at position i. For traces $t_1, \dots, t_n \in \Sigma^\omega$ we define $zip(t_1, \dots, t_n) \in (\Sigma^n)^\omega$ as the pointwise product of the traces, i.e., $zip(t_1, \dots, t_n)(i) := (t_1(i), \dots, t_n(i))$. We occasionally write $t_1 \otimes t_2$ instead of $zip(t_1, t_2)$.

1) Transition Systems: A transition system is a tuple $\mathcal{T} = (S, S_0, \varrho, L)$ where S is a finite set of states, $S_0 \subseteq S$ a set of initial states, $\varrho \subseteq S \times S$ a transition relation, and $L : S \to \Sigma$ a state labelling. We write $s \xrightarrow{\mathcal{T}} s'$ whenever $(s, s') \in \varrho$ and define $Sucs(s) := \{s' \mid (s, s') \in \varrho\}$. We assume $Sucs(s) \neq \emptyset$ for every $s \in S$. A path in $\mathcal{T}$ is an infinite sequence $p \in S^\omega$ such that $p(0) \in S_0$ and for every $i \in \mathbb{N}$, we have $p(i + 1) \in Sucs(p(i))$. Each path p denotes a trace $L(p) \in \Sigma^\omega$ by applying the labelling pointwise, i.e., $L(p)(i) := L(p(i))$. We write $Paths(\mathcal{T})$ for the set of all paths and $Traces(\mathcal{T})$ for the set of all traces. For $s \in S$ we define $\mathcal{T}_s$ as the transition systems obtained by changing the initial states to $\{s\}$.

2) ω-Automata: A deterministic ω-automaton over alphabet Σ is a tuple $\mathcal{A} = (Q, q_0, \delta, Acc)$ where Q is a finite set of states, $q_0 \in Q$ an initial state, $\delta : Q \times \Sigma \to Q$ a transition function, and $Acc \subseteq Q^\omega$ the acceptance condition. For every finite word $u \in \Sigma^*$, we define $\delta^*(u) \in Q$ as the unique state reached when reading u (starting in q_0). For a trace $t \in \Sigma^\omega$, the unique run $r_t \in Q^\omega$ is given by $r_t(i) := \delta^*(t[0, i-1])$ where $t[0, i-1]$ is the prefix of length i. We write $\mathcal{L}(\mathcal{A})$ for the language of the automaton, which consists of all traces t whose unique run r_t satisfies $r_t \in Acc$. In a Büchi automaton, the acceptance is given by a set $F \subseteq Q$ of accepting states, and a run is accepting if it visits states in F infinity many times. In a parity automaton, the acceptance is given by a coloring $c : Q \to \mathbb{N}$, and a run is accepting if the minimal color occurring infinitely often (as given by c) is even. In a safety automaton, the acceptance is given by a set $B \subseteq Q$ of bad states, and a run is accepting if it never visits a state in B. A language $\mathcal{L} \subseteq \Sigma^\omega$ is ω-regular if there exists a deterministic parity automaton (DPA) that recognizes it.[4] A language $\mathcal{L} \subseteq \Sigma^\omega$ is

[3] In particular, the completeness proof in [8] is purely semantic. The history and prophecy variables describe the past and future behavior of the system, which, in the worst case, turns a finite-state system into an infinite-state one.

[4] Throughout this paper, we work with deterministic ω-automata. Any non-deterministic Büchi automaton (NBA) (see, e.g., [21] for a formal definition) can be effectively translated into a DPA [22], [23]. On the other hand, deterministic Büchi and deterministic safety automata are strictly less expressive and do not capture full ω-regularity.

safety [24], [25], if it can be recognized by a deterministic safety automata. Given $q \in Q$ we define $\mathcal{A}_q$ as the automaton obtained by replacing the initial state with q. For a set $X \subseteq Q$ and a trace $t \in \Sigma^\omega$, we define $\mathit{firstVisit}_X(\mathcal{A}, t) \in \mathbb{N} \cup \{\infty\}$ as the first time step where the unique run of $\mathcal{A}$ on t visits a state in X (if it exists and ∞ otherwise).

3) Parity Games: A parity game is a tuple $\mathcal{G} = (V_{\mathfrak{V}}, V_{\mathfrak{R}}, T, c)$ where $V := V_{\mathfrak{V}} \uplus V_{\mathfrak{R}}$ is the finite set of states. The states in $V_{\mathfrak{V}}$ are controlled by the verifier $\mathfrak{V}$ and those in $V_{\mathfrak{R}}$ are controlled by the refuter $\mathfrak{R}$. $T \subseteq V \times V$ is the transition relation (we assume that for each v there is at least one v' with $(v, v') \in T$), and $c : V \to \mathbb{N}$ the coloring of each node. A strategy σ for player $p \in \{\mathfrak{V}, \mathfrak{R}\}$ is a function $\sigma : V^* \times V_p \to V$ such that for every $\mathbf{v} \in V^*, v \in V_p$, $(v, \sigma(\mathbf{v}, v)) \in T$. A play in $\mathcal{G}$ is an infinite sequence $r \in V^\omega$ such that for every i, $(r(i), r(i+1)) \in T$. The play r is compatible with strategy σ for player p if for every i where $r(i) \in V_p$ we have that $r(i+1) = \sigma(r(0) \cdots r(i-1), r(i))$. A play r is won by player $\mathfrak{V}$ if the minimal color occurring infinitely often in r (according to c) is even. Otherwise, it is won by $\mathfrak{R}$. We say that player p wins node v if there exists a strategy σ for p such that every play that starts in v and is compatible with σ is won by p. As parity games are positionally determined [26], every node is either won by $\mathfrak{V}$ or by $\mathfrak{R}$.

4) HyperLTL: As the basic specification language for hyperproperties we use HyperLTL [2], which extends linear-time temporal logic (LTL) with explicit trace quantification. We assume a fixed set of trace variables $\mathcal{V}$. Formulas in HyperLTL are generated by the following grammar.

$$\begin{aligned} \varphi &:= \exists \pi . \varphi \mid \forall \pi . \varphi \mid \phi \\ \phi &:= a_\pi \mid \neg \phi \mid \phi_1 \wedge \phi_2 \mid \bigcirc \phi \mid \phi_1 \mathcal{U} \phi_2 \end{aligned}$$

where $\pi \in \mathcal{V}$ and $a \in AP$. We use the derived boolean connectives $\vee, \rightarrow, \leftrightarrow$, boolean constants $\top, \bot$, and temporal operators eventually ($\diamondsuit \phi := \top \mathcal{U} \phi$) and globally ($\square \phi := \neg \diamondsuit \neg \phi$). We consider only closed formulas, i.e., formulas where for each atom a_π the trace variable π is bound by some trace quantifier. The semantics of HyperLTL is given with respect to a set of traces $\mathbb{T} \subseteq \Sigma^\omega$ and a trace assignment Π, which is a partial mapping $\Pi : \mathcal{V} \rightharpoonup \Sigma^\omega$. For $\pi \in \mathcal{V}$ and trace t, we write $\Pi[\pi \mapsto t]$ for the trace assignment obtained by updating the value of π to t.

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

We say a transition system $\mathcal{T}$ satisfies φ, written $\mathcal{T} \models \varphi$, if $\emptyset \models_{\mathit{Traces}(\mathcal{T})} \varphi$ where $\emptyset$ denotes the empty trace assignment.

5) Quantified Propositional Temporal Logic (QPTL): The prophecies we study in this paper are ω-regular sets. LTL is limited to non-counting properties and can consequently not express arbitrary ω-regular properties [27]. To nevertheless support prophecies on a syntactic level (where we represent prophecies as formulas instead of ω-automata), we use Quantified Propositional Temporal Logic (QPTL) [28]. We assume a fresh set of propositional variables PV. We define QPTL formulas by the following grammar.

$$\phi := a_\pi \mid \neg \phi \mid \phi_1 \wedge \phi_2 \mid \bigcirc \phi \mid \phi_1 \mathcal{U} \phi_2 \mid \tilde{\exists} q . \phi \mid q$$

where $\pi \in \mathcal{V}$, $a \in AP$ and $q \in PV$. QPTL allows the quantification of a proposition variable q using $\tilde{\exists} q . \phi$ and to refer to the truth value of each propositional variable. We abbreviate $\tilde{\forall} q . \phi := \neg \tilde{\exists} q . \neg \phi$. Note that we write $\tilde{\exists}$ and $\tilde{\forall}$ for propositional quantification to visually distinguish them from the trace quantifiers in HyperLTL. The semantics of QPTL is defined similarly to before with an additional mapping $\Delta : PV \rightharpoonup \mathbb{B}^\omega$ that handles propositional quantification (where $\mathbb{B} = \{\top, \bot\}$).

$$\begin{array}{lll} \Pi, \Delta, i \models a_\pi & \text{iff} & a \in \Pi(\pi)(i) \\ \Pi, \Delta, i \models \neg \phi & \text{iff} & \Pi, \Delta, i \not\models \phi \\ \Pi, \Delta, i \models \phi_1 \wedge \phi_2 & \text{iff} & \Pi, \Delta, i \models \phi_1 \text{ and } \Pi, \Delta, i \models \phi_2 \\ \Pi, \Delta, i \models \bigcirc \phi & \text{iff} & \Pi, \Delta, i+1 \models \phi \\ \Pi, \Delta, i \models \phi_1 \mathcal{U} \phi_2 & \text{iff} & \exists j \geq i . \Pi, \Delta, j \models \phi_2 \text{ and} \\ & & \quad \forall i \leq k < j . \Pi, \Delta, k \models \phi_1 \\ \Pi, \Delta, i \models \tilde{\exists} q . \phi & \text{iff} & \exists \tau \in \mathbb{B}^\omega . \Pi, \Delta[q \mapsto \tau], i \models \varphi \\ \Pi, \Delta, i \models q & \text{iff} & \Delta(q)(i) = \top \end{array}$$

The main advantage of QPTL (over LTL) stems from the following result:

Theorem 1 ([28])**.** *A language $\mathcal{L}$ is ω-regular if and only if it is definable in QPTL.*

Example 1. *Take the property "a holds on trace π in at least one even position". While not expressible in LTL [27], we can express it in QPTL as*

$$\tilde{\exists} q . \, q \wedge \square(q \leftrightarrow \bigcirc \neg q) \wedge \diamondsuit(a_\pi \wedge q).$$

In the remainder of this paper, we assume no particular familiarity with QPTL and only use it when absolutely necessary. We resort to QPTL as a tool to express ω-regular properties as formulas which allows us to treat prophecies at a syntactic level. Our prophecy construction itself is language-theoretic.

V. Strategy-based Verification

The problem we are tackling in this paper is the following: Given a transition system $\mathcal{T}$ and a $\forall^*\exists^*$-HyperLTL property φ, check if $\mathcal{T} \models \varphi$. A first practical verification approach was proposed by Coenen et al. [7], which we refer to as *strategy-based verification*. The idea is to instantiate existential quantification with a strategy that incrementally constructs a trace by reacting to the moves of the $\forall$-player. Coenen

$$\frac{s_1 \xrightarrow{\mathcal{T}} s_1' \qquad \cdots \qquad s_k \xrightarrow{\mathcal{T}} s_k' \qquad q' = \delta^{\phi}\Big(q, \big(L(s_1), \cdots, L(s_{k+l})\big)\Big)}{\langle (s_1, \dots, s_k, s_{k+1}, \dots, s_{k+l}), q, \forall \rangle \to \langle (s_1', \dots, s_k', s_{k+1}, \dots, s_{k+l}), q', \exists \rangle}\ (\forall)$$

$$\frac{s_{k+1} \xrightarrow{\mathcal{T}} s_{k+1}' \qquad \cdots \qquad s_{k+l} \xrightarrow{\mathcal{T}} s_{k+l}'}{\langle (s_1, \dots, s_k, s_{k+1}, \dots, s_{k+l}), q, \exists \rangle \to \langle (s_1, \dots, s_k, s_{k+1}', \dots, s_{k+l}'), q, \forall \rangle}\ (\exists)$$

$$\frac{s_{k+1} \in S_0 \qquad \cdots \qquad s_{k+l} \in S_0}{(s_1, \dots, s_k) \to \langle (s_1, \dots, s_k, s_{k+1}, \dots, s_{k+l}), q_0^{\phi}, \forall \rangle}\ \text{(init)}$$

Fig. 3: Transition rules for the parity-game-based synthesis of winning strategies for the $\exists$-player.

et al. formalize the strategy as a finite state transducer that determines the next move of all existentially quantified copies. The automated synthesis of a strategy is then expressed as a SMT constraints. We phrase the problem as a parity game which serves as an easier formal foundation to discuss our completeness results.

A. Strategy-based Verification as a Parity Game

The idea is that the parity game mimic the iterative trace construction of both players. Assume we are given a system $\mathcal{T} = (S, S_0, \varrho, L)$ and a HyperLTL formula

$$\varphi = \forall \pi_1 \dots \pi_k . \exists \pi_{k+1} \dots \pi_{k+l} . \, \phi.$$

We define a parity game $\mathcal{G}_{\mathcal{T},\varphi}$ as follows. Let $\mathcal{A}^{\phi} = (Q^{\phi}, q_0^{\phi}, \delta^{\phi}, c^{\phi})$ be a deterministic parity automaton (DPA) over Σ^{k+l} for ϕ that accepts exactly the zippings of traces that satisfy the formula, i.e., $[\pi_1 \mapsto t_1, \dots, \pi_{k+l} \mapsto t_{k+l}] \models \phi$ if and only if $\mathit{zip}(t_1, \dots, t_{k+l}) \in \mathcal{L}(\mathcal{A}^{\phi})$. The construction of this automaton can be performed via a standard LTL to DPA translation (see, e.g., [6], [29]).

The game $\mathcal{G}_{\mathcal{T},\varphi}$ comprises two node kinds: Nodes are either of the form $(s_1, \dots, s_k)$ where $s_i \in S$ for all $1 \leq i \leq k$ to encode the initial states of the universally quantified copies. Or they are of the form $\langle (s_1, \dots, s_{k+l}), q, \flat \rangle$ where $s_i \in S$ for all $1 \leq i \leq k+l$, $q \in Q^{\phi}$ and $\flat \in \{\forall, \exists\}$. Here $(s_1, \dots, s_{k+l})$ gives the current state of all copies of $\mathcal{T}$, q is the current state of the DPA tracking ϕ, and $\flat$ defines whether the universal ($\flat = \forall$) or existential ($\flat = \exists$) copies move next. Nodes of the form $\langle (s_1, \dots, s_{k+l}), q, \forall \rangle$ are controlled by the refuter (who takes the role of the $\forall$-player), and nodes of the form $\langle (s_1, \dots, s_{k+l}), q, \exists \rangle$ and $(s_1, \dots, s_k)$ are controlled by the verifier (who takes the role of the $\exists$-player). The transitions of the game are given in Figure 3. The ($\forall$) and ($\exists$)-transition rules are the game's main rules. In the ($\forall$)-rule all universally quantified copies are updated by moving to successor states within $\mathcal{T}$. Simultaneously, we update the automaton state of $\mathcal{A}^{\phi}$. Similarly, in the ($\exists$)-rule, the existentially quantified copies are updated. The (init)-rule is used at the beginning where the universal copies have already chosen a state and the existential copies can select any initial state for themself. Lastly, the coloring of the nodes is obtained by assigning each node of the form $\langle (s_1, \dots, s_{k+l}), q, \flat \rangle$ the color given by $c^{\phi}(q)$. The color of nodes of the form $(s_1, \dots, s_k)$ is irrelevant as they are visited at most once.

B. Soundness of Strategy-based Verification

The game $\mathcal{G}_{\mathcal{T},\varphi}$ mimics the strategic behavior of the $\exists$-player. In each step, the refuter chooses successors for the k universally quantified traces, followed by the verifier who selects successors for the l existentially quantified traces. The automaton state in the nodes of $\mathcal{G}_{\mathcal{T},\varphi}$ tracks the (unique) run of $\mathcal{A}^{\phi}$ on the resulting $k+l$ traces. To verify that $\mathcal{T} \models \varphi$, the verifier should win from every possible combination of initial states for the universally quantified copies. We define

$$V_{\mathit{init}} \coloneqq \{(s_1, \dots, s_k) \mid \forall 1 \leq i \leq k . \, s_i \in S_0\}.$$

We write $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T},\varphi}$ if the verifier wins $\mathcal{G}_{\mathcal{T},\varphi}$ from all nodes in V_{init}. We can show the soundness of our verification method.

Theorem 2. *If $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T},\varphi}$ then $\mathcal{T} \models \varphi$.*

Proof sketch. We use a positional winning strategy σ for $\mathfrak{V}$ that witnesses $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T},\varphi}$ to iteratively construct traces for the existentially quantified traces by simulating σ on finite prefixes of the universally quantified traces. We give a detailed proof in the full version [30]. □

C. Prophecies and Prophecy Variables

As we saw in Section II-A, strategy-based verification of $\forall^*\exists^*$ properties is incomplete, i.e., $\mathfrak{V}$ might lose $\mathcal{G}_{\mathcal{T},\varphi}$ even though the system satisfies the property. Intuitively, this is the case when the $\exists$-player (the verifier in $\mathcal{G}_{\mathcal{T},\varphi}$) needs future information that is not available by observing only a prefix of the universally quantified traces. To counteract this lack of information, we introduce prophecies.

Definition 1. *A* prophecy *is a ω-regular subset $\mathfrak{P} \subseteq (\Sigma^k)^{\omega}$.*

If a prophecy $\mathfrak{P}$ holds at step i, the $\exists$-player can assume that the $\forall$-player (the refuter in $\mathcal{G}_{\mathcal{T},\varphi}$) starting in step i, constructs traces $t_1, \dots, t_k$ for the k universal quantifiers such that $\mathit{zip}(t_1, \dots, t_k) \in \mathfrak{P}$. The prophecy thereby provides limited information (in form of the binary information on whether or not the prophecy holds) about the future behavior of the universally quantified traces.

To formally introduce prophecies into our framework, we need to enable the $\exists$-player to, in each step, determine which prophecies hold. We delegate this step to the universal player who determines the truth value for each prophecy in its (modified) state space. Formally, we accomplish this in two steps. (1) We extend the system by fresh boolean variables (called *prophecy variables*) that, in each step, can be chosen non-deterministically, and (2) we relax the specification to ensure that the prophecy variables set by the $\forall$-player correspond to the truth value of the prophecies.

1) System Manipulation: We begin by modifying the transition system to allow the $\forall$-player to set the prophecy variables.

Definition 2. *Given a transition system $\mathcal{T} = (S, S_0, \varrho, L)$ and a set of fresh propositions P (with $P \cap AP = \emptyset$) we define the modified transition system $\mathcal{T}^P := (S^P, S_0^P, \varrho^P, L^P)$ over $AP \uplus P$ where $S^P := S \times 2^P$, $S_0^P := S_0 \times 2^P$, $\varrho^P := \{((s, A), (s', A')) \mid (s, s') \in \varrho \wedge A, A' \in 2^P\}$ and $L^P(s, A) := L(s) \cup A$.*

In particular, we have

$$\mathit{Traces}(\mathcal{T}^P) = \{t \cup t' \mid t \in \mathit{Traces}(\mathcal{T}), t' \in (2^P)^\omega\}$$

where $t \cup t'$ denotes the pointwise union of both traces.

2) Property Manipulation: We modify the matrix of the hyperproperty such that the original property is only required to hold, if all prophecies by the universal player are set correctly, i.e., a prophecy variable in P is set to true iff the universally quantified traces produced by the $\forall$-player are contained in the corresponding prophecy. To express this at the logical level, we make use of the fact that we can express a (ω-regular) prophecy $\mathfrak{P}$ as a QPTL formula (cf. Theorem 1).[5]

Definition 3. *Given a set of QPTL formulas $\Xi = \{\xi_1, \dots, \xi_n\}$ using only trace variables in $\{\pi_1, \dots, \pi_k\}$ and a fresh set of atomic propositions $P = \{p_1, \dots, p_n\}$, define the modified formula $\varphi^{P,\Xi}$ as*

$$\forall \pi_1 \dots \forall \pi_k . \exists \pi_{k+1} \dots \exists \pi_{k+l} . \left[\square \bigwedge_{j=1}^{n} (p_{j_{\pi_1}} \leftrightarrow \xi_j) \right] \to \phi.$$

That is, we only require ϕ to hold, if in every step and for every $1 \leq j \leq n$, the prophecy formula ξ_j holds exactly when the prophecy variable p_j is set on trace π_1.[6]

3) Soundness of Prophecies: The combination of the modified transition system (which allows the prophecy variables to take any value) and the modified property does not impact the satisfaction of the original property on the original system as stated in the following theorem (see, e.g., [7, Thm. 5]).

Theorem 3. *Let $\Xi = \{\xi_1, \dots, \xi_n\}$ and $P = \{p_1, \dots, p_n\}$ be as in Definition 3. Then $\mathcal{T} \models \varphi$ if and only if $\mathcal{T}^P \models \varphi^{P,\Xi}$.*

Remark 1. *A brief remark about nomenclature is in order. A* prophecy *is a ω-regular set of traces $\mathfrak{P}$. We represent this prophecy as a QPTL formula $\xi_i \in \Xi$ which we also refer to as a prophecy or* prophecy formula. *Lastly, $p_i \in P$ is a* prophecy variable *that corresponds to prophecy (formula) ξ_i.*

4) Prophecies for Strategy-based Verification: While the addition of prophecies does not alter the satisfaction of the property in the HyperLTL semantics (as stated in Theorem 3), it can impact the existence of a winning strategy for the $\exists$-player during strategy-based verification. That is, it might be that $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T},\varphi}$ does not hold, but $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}^P,\varphi^{P,\Xi}}$ does. Thus, prophecies provide a natural tool to strengthen strategy-based verification and allow the user to, e.g., introduce domain knowledge in the form of user-defined prophecies. The soundness of the addition of prophecies can be argued easily: If $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}^P,\varphi^{P,\Xi}}$ holds, then (by Theorem 2) $\mathcal{T}^P \models \varphi^{P,\Xi}$ so (by Theorem 3) $\mathcal{T} \models \varphi$. The situation is depicted graphically in Figure 4a.

Example 2. *With our notation fixed, we revisit the transition system $\mathcal{T}$ and HyperLTL formula φ from Section II-A. In this case, $\mathfrak{V} \not\vdash \mathcal{G}_{\mathcal{T},\varphi}$, i.e., strategy-based verification fails without the addition of prophecies. Let $\Xi = \{\bigcirc a_\pi\}$ and let $P = \{p\}$ be a fresh set of prophecies variables. Using Definition 3 we construct*

$$\varphi^{P,\Xi} = \forall \pi . \exists \pi' . \square (p_\pi \leftrightarrow \bigcirc a_\pi) \to \square (a_{\pi'} \leftrightarrow \bigcirc a_\pi).$$

It is easy to see that $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}^P,\varphi^{P,\Xi}}$: the prophecy variable p hints at the next move of $\mathfrak{R}$. If, for example, $\mathfrak{R}$ sets p to true, $\mathfrak{V}$ can assume that $\bigcirc a_\pi$ holds (if it does not, the premise of $\varphi^{P,\Xi}$ is violated and so the play is trivially won by $\mathfrak{V}$). The verifier can thus move to state s_1 (in Figure 1a) and thereby correctly predict the next move on π. As argued in Figure 4a, $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}^P,\varphi^{P,\Xi}}$ implies that $\mathcal{T} \models \varphi$.

VI. Completeness

We have argued that strategy-based verification remains sound when adding prophecies. The natural question that arises is the following:

> *Assume that $\mathcal{T} \models \varphi$. Does there exist* some *finite set of prophecies Ξ such that $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}^P,\varphi^{P,\Xi}}$?*

As already observed by Coenen et al. [7], this does not hold if we only allow LTL-definable prophecies.

Example 3. *Consider a system $\mathcal{T}$ that generates all traces over $AP = \{a, b\}$ and the following property φ*

$$\forall \pi . \exists \pi' . a_{\pi'} \wedge \square (a_{\pi'} \leftrightarrow \bigcirc \neg a_{\pi'}) \wedge (b_{\pi'} \leftrightarrow \diamondsuit (b_\pi \wedge a_{\pi'})).$$

That is, b should hold in the first step on π' iff b holds at some even *position on π. Clearly, $\mathcal{T} \models \varphi$ but $\mathfrak{V} \not\vdash \mathcal{G}_{\mathcal{T},\varphi}$. Moreover, LTL cannot express that (on π) b ever holds at an even position (cf. Example 1). Consequently, no LTL-definable*

[5] In practice, we would not express prophecies in QPTL and instead operate directly on an automaton-based representation of a prophecy. By taking this detour, we can keep the notation succinct and can express the assumption that the $\forall$-player correctly sets the prophecy variables as a logical implication.

[6] With the construction of $\varphi^{P,\Xi}$ we ensure that each prophecy variable on π_1 reflects the truth value of the prophecy. However, any of the universally quantified trace variables would work equally well. Note that each prophecy formula ξ_j captures a behavior of the combined executions of the universally quantified traces $\pi_1, \dots, \pi_k$ (as ξ_j uses trace variables in $\{\pi_1, \dots, \pi_k\}$) and not necessarily the behavior of a single trace. In fact, local prophecies (i.e., prophecies that only capture behavior on one trace) are insufficient for completeness (cf. Example 4).

$$\begin{tikzcd}
 & \mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Xi}} \arrow[d, "\text{Thm. 2}"] \\
\mathcal{T} \models \varphi \arrow[r, shift left] & \mathcal{T}^P \models \varphi^{P,\Xi} \arrow[l, shift left, "\text{Thm. 3}"]
\end{tikzcd}$$

(a) Implications for any set of prophecies.

$$\begin{tikzcd}
 & \mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Xi}} \arrow[d, "\text{Thm. 2}"] \\
\mathcal{T} \models \varphi \arrow[ur, "\text{Thm. 4}"] \arrow[r, shift left] & \mathcal{T}^P \models \varphi^{P,\Xi} \arrow[l, shift left, "\text{Thm. 3}"]
\end{tikzcd}$$

(b) Implications for a *complete* set of prophecies.

Fig. 4: Implications between the satisfaction of a hyperproperty and the existence of a winning strategy for the $\exists$-player. We display implications with an arbitrary set of prophecies (Figure 4a) and a complete set (Figure 4b).

prophecy can provide sufficient information to the $\exists$-player, i.e., $\mathfrak{V} \not\vdash \mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Xi}}$ for any (finite) set of LTL formulas Ξ.

While this incompleteness result for LTL-definable prophecies is interesting in its own right, we usually do not represent prophecies as LTL formulas but work with some automaton representation. Consequently, we are less interested in LTL-definable prophecies but in the existence of ω-regular prophecies. And indeed, in this paper, we show that we can answer the above question positively if we shift from LTL-definable prophecies to arbitrary ω-regular prophecies. The main result of this paper reads as follows:

Theorem 4. *Let $\mathcal{T}$ be a (finite-state) transition system and let φ be a $\forall^*\exists^*$ HyperLTL property such that $\mathcal{T} \models \varphi$. There exist finitely many QPTL prophecies $\Xi = \{\xi_1, \dots, \xi_n\}$ such that for a fresh set $P = \{p_1, \dots, p_n\}$ we get $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Xi}}$.*

If $\mathcal{T} \models \varphi$ we call a set of prophecies Ξ *complete* if $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Xi}}$, i.e., Ξ is a witness to Theorem 4. The resulting situation is depicted in Figure 4b. Combined with Theorem 2 and Theorem 3 we can rephrase Theorem 4 as follows:

Corollary 1. *Let $\mathcal{T}$ be a (finite-state) transition system and let φ be a $\forall^*\exists^*$ HyperLTL property. There exist finitely many QPTL prophecies $\Xi = \{\xi_1, \dots, \xi_n\}$ such that for a fresh set $P = \{p_1, \dots, p_n\}$ we get $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Xi}}$ if and only if $\mathcal{T} \models \varphi$.*

We note that our prophecy construction used to prove Theorem 4 yields prophecies *without* first checking if $\mathcal{T} \models \varphi$. This allows us to use our construction to (algorithmically) check if $\mathcal{T} \models \varphi$ (we discuss this in Section IX-A).

Remark 2. *We can strengthen Theorem 4 further. Our prophecy construction treats the LTL matrix of the HyperLTL property as an ω-automaton. The constructions thus generalize to all logics that utilize the trace quantification mechanism of HyperLTL but express arbitrary ω-regular property within their matrix. For example, our result also applies to HyperQPTL, i.e., formulas where the trace-quantifier prefix is followed by a QPTL formula. We thus show that ω-regular prophecies suffice for all $\forall^*\exists^*$ hyperproperties with ω-regular matrix. In contrast, Example 3 shows that LTL-definable prophecies are not sufficient for $\forall^*\exists^*$ hyperproperties with LTL-definable matrix (aka. HyperLTL).*

Example 4. *We can show that in the case of more than a single universally quantified trace (i.e., cases where $k > 1$), prophecies must necessarily reason about the joint future behavior of all k universally quantified traces. Consider the transition system $\mathcal{T}$ in Figure 1a that generates all traces over $AP = \{a\}$ and the property*

$$\varphi = \forall \pi. \forall \pi'. \exists \pi''.\, a_{\pi''} \leftrightarrow \Box(a_\pi \leftrightarrow a_{\pi'}).$$

That is, a should hold on π'' in the first step iff π and π' are equal. Clearly, $\mathcal{T} \models \varphi$ but this cannot be verified using strategy-based verification without prophecies. The LTL-definable prophecy $\xi := \Box(a_\pi \leftrightarrow a_{\pi'})$ provides enough information to the $\exists$-player on whether or not to set a in the first step. However, any finite set of local prophecies (i.e., prophecy formulas that only refer to π or only refer to π') is incomplete.

The following two sections are devoted to a proof of Theorem 4. To avoid clustered notation, we give our proof for hyperproperties of the form $\forall \pi. \exists \pi'. \phi$. Our result generalizes easily to the entire $\forall^*\exists^*$ fragment. We begin our proof by considering HyperLTL formulas of the form $\forall \pi. \exists \pi'. \phi$ where ϕ, when interpreted as a trace property, is a *safety* property (in the traditional sense [24]). This allows for a simpler construction (in Section VII). In Section VIII we then incrementally extend the construction to general temporal properties.

Remark 3. *It is important to note that the class of safety used in Section VII only refers to the LTL matrix (the body) of the HyperLTL property. If the matrix is safety, this does* not *imply that the HyperLTL formula is hypersafety (as defined by Clarkson and Schneider [1]). For example, the matrix of GNI (cf. Section I) is a safety property (and thus lends itself to the simpler construction in Section VII), but GNI is a hyperliveness property [1], [7]. On the other hand, as shown in [31], the class of formulas with safety matrix (called* temporal safety *in [31]) already contains all $\forall^*\exists^*$ hypersafety properties.*

VII. Completeness for Safety Matrix

We first consider the case where ϕ is a safety property. Let $\mathcal{A}^\phi = (Q^\phi, q_0^\phi, \delta^\phi, B^\phi)$ be a deterministic safety automaton over $\Sigma \times \Sigma$ for ϕ.

A. Prophecy Construction

The main idea behind our completeness result (which in a modified form also applies to the general case in Section VIII) is to design prophecies that directly identify those states that the $\exists$-player should move to. As we assume that ϕ denotes a safety property, we can accomplish this by identifying all states that are *safe*, i.e., all states that the $\exists$-player can move to without losing the game immediately. Formally, we add a prophecy for each state s of the game and design them such that a trace constructed by the $\forall$-player lies within a prophecy for state s if and only if choosing s as a successor is safe for the $\exists$-player. For every $q \in Q^\phi$ and $s \in S$ we define

$$\mathfrak{P}_{q,s} := \{t \in \Sigma^\omega \mid \exists t' \in \mathit{Traces}(\mathcal{T}_s).\, t \otimes t' \in \mathcal{L}(\mathcal{A}^\phi_q)\}.$$

Recall that $\mathcal{T}_s$ is $\mathcal{T}$ with s fixed as the initial state and similarly for $\mathcal{A}^\phi_q$. That is, a trace t (chosen for the universally quantified trace in φ) is in $\mathfrak{P}_{q,s}$ if there exists some trace (chosen for the existentially quantified trace in φ) that starts in s and, in combination with t, is accepted by $\mathcal{A}^\phi$ (when starting in q).

To (informally) see why these prophecies are useful for the $\exists$-player, let us assume that the current state of $\mathcal{A}^\phi$ (on the current prefix of the game) is q. If prophecy $\mathfrak{P}_{q,s}$ holds, the $\exists$-player can move to state s knowing that the $\forall$-player plays such that s is a safe move (as *some* trace starting from s is still winning).

Remark 4. *In our formalization, the $\forall$-player sets the prophecy variables. Conceptually, we can thus consider prophecies as a binding contract between the $\forall$-player and the $\exists$-player. When the $\forall$-player indicates that $\mathfrak{P}_{q,s}$ holds (by setting the respective prophecy variable), the $\forall$-player enters a binding agreement that guarantees that the constructed trace is contained in $\mathfrak{P}_{q,s}$ (as otherwise, the premise of $\varphi^{P,\Xi}$ is violated so the $\exists$-player wins trivially). From this point of view, our prophecies defer the selection of a successor state from the $\exists$-player to the $\forall$-player: By setting the variables, the $\forall$-player implicitly fixes all valid moves for the $\exists$-player.*

We can easily see that the resulting prophecies are ω-regular (by constructing the product of $\mathcal{T}_s$ and $\mathcal{A}^\phi_q$). Consequently, we can represent each prophecy $\mathfrak{P}_{q,s}$ as a QPTL prophecy formula $\xi_{q,s}$ (cf. Theorem 1). The resulting set of prophecies is complete in the sense of Theorem 4.

Theorem 5. *Assume $\mathcal{T} \models \varphi$. Define $\Xi = \{\xi_{q,s} \mid q \in Q^\phi, s \in S\}$ and let $P = \{p_{q,s} \mid q \in Q^\phi, s \in S\}$ be a fresh set of atomic propositions. Then $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Xi}}$.*

B. Correctness Proof

In this subsection, we sketch a proof of Theorem 5. As a complete proof is rather involved, we restrict ourselves to the construction of a winning strategy for the $\exists$-player and refer to a detailed proof in the full version [30]. Readers less interested in the proof can skip to Section VII-C.

1) Notation: We begin by introducing some notation. By definition of $\mathcal{T}^P$, nodes in $\mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Xi}}$ either have the form (s, A), where $s \in S$ and $A \subseteq P$ or the form $\langle (s, A), (s', A'), q, \flat \rangle$, where $s, s' \in S$, $A, A' \subseteq P$ and $\flat \in \{\forall, \exists\}$. Here q is an automaton state in a DPA tracking

$$\phi^{P,\Xi} := \square\Big(\bigwedge_{q \in Q^\phi, s \in S} (p_{q,s})_\pi \leftrightarrow \xi_{q,s}\Big) \to \phi. \tag{2}$$

It is easy to see that in states of the form $\langle (s, A), (s', A'), q, \flat \rangle$ the A' component (stemming from the definition of $\mathcal{T}^P$) is irrelevant as in (2) the prophecy variables are only referred to on trace variable π. We, therefore, consider a node $\langle (s, A), (s', A'), q, \flat \rangle$ simply as $\langle (s, A), s', q, \flat \rangle$. With this conceptual simplification, any finite play $\mathbf{v} \in V^*$ in $\mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Xi}}$ (starting in some state in V_{init}) of odd-length (where $|\mathbf{v}| = 2i + 1$) has the form

$$\begin{aligned}&(s_0, A_0) \to \langle (s_0, A_0), s'_0, q_0, \forall \rangle \to \langle (s_1, A_1), s'_0, q_1, \exists \rangle \\ &\to \langle (s_1, A_1), s'_1, q_1, \forall \rangle \to \cdots \to \langle (s_i, A_i), s'_{i-1}, q_i, \exists \rangle.\end{aligned} \tag{3}$$

We can extract from $\mathbf{v}$ both paths through $\mathcal{T}$ and the prophecy variables set at each step. Define $s_\mathbf{v}(0)s_\mathbf{v}(1) \cdots s_\mathbf{v}(i)$ to be the path of the $\forall$-player ($s_0 s_1 \cdots s_i$ in (3)), $A_\mathbf{v}(0)A_\mathbf{v}(1) \cdots A_\mathbf{v}(i)$ the sequence of prophecy variables chosen ($A_0 A_1 \cdots A_i$ in (3)), and $s'_\mathbf{v}(0)s'_\mathbf{v}(1) \cdots s'_\mathbf{v}(i-1)$ the path for the $\exists$-player ($s'_0 s'_1 \cdots s'_{i-1}$ in (3)). Define $t_\mathbf{v}(k) := L(s_\mathbf{v}(k))$ and $t'_\mathbf{v}(k) := L(s'_\mathbf{v}(k))$ for $0 \leq k \leq i - 1$.

2) Strategy Construction: With those definitions at hand, we define an explicit winning strategy σ for $\mathfrak{V}$ as follows:

```
 1: Input: $\mathbf{v} \in V^*$ with $|\mathbf{v}| = 2i + 1$
 2: if $i = 0$ then
 3:     $T := S_0$
 4: else
 5:     $T := \mathit{Sucs}(s'_\mathbf{v}(i-1))$
 6: $\hat{q} := {\delta^\phi}^*\big[(t_\mathbf{v}(0), t'_\mathbf{v}(0)) \cdots (t_\mathbf{v}(i-1), t'_\mathbf{v}(i-1))\big]$
 7: $C := \{s' \mid s' \in T \land p_{\hat{q},s'} \in A_\mathbf{v}(i)\}$
 8: if $C \neq \emptyset$ then
 9:     return any $s' \in C$
10: else
11:     return any $s' \in T$
```

Note that σ directly returns a successor state in $\mathcal{T}$.

By the structure of $\mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Xi}}$, any finite path starting in V_{init} that reaches a node in $V_\mathfrak{V}$ is of odd length. We begin by computing all possible successor states for the $\exists$-player in a set T. These are either all initial nodes in the case where $|\mathbf{v}| = 1$ (line 3) or all successor states of the current state of the $\exists$-player (line 5). We then compute the state $\hat{q}$ of $\mathcal{A}^\phi$ reached on $\mathbf{v}$ in line 6. Note that $\hat{q}$ is a state in $\mathcal{A}^\phi$ whereas the automaton states occurring in $\mathbf{v}$ are states in a DPA tracking (2). In line 7, we check if any of the possible successors in T are declared safe by the $\forall$-player, i.e., we check for states where the corresponding prophecy variable is set. If there is any such state, we pick it (line 9). Otherwise, we choose an arbitrary successor (line 11).

Example 5. *We can simulate the strategy on abstract prefixes of (3). Initially, for $\mathbf{v} = (s_0, A_0)$ it picks any initial state $s'_0 \in S_0$ such that $p_{s'_0, q^\phi_0} \in A_0$. For path $\mathbf{v} = (s_0, A_0) \to \langle (s_0, A_0), s'_0, q_0, \forall \rangle \to \langle (s_1, A_1), s'_0, q_1, \exists \rangle$ it computes the current state $\hat{q}$ of $\mathcal{A}^\phi$ reached on the path $(L(s_0), L(s'_0)) \in (\Sigma \times \Sigma)^*$ and picks any successor s'_1 of s'_0 such that $p_{s'_1, \hat{q}} \in A_1$.*

It remains to argue the correctness of the just constructed strategy. Here, we may assume that all prophecies are set correctly (i.e., the premise of (2) is true) as otherwise, the play is trivially won by $\mathfrak{V}$. Under this assumption, the premise that $\mathcal{T} \models \varphi$, and by induction on the length of a prefix of (3) we can establish that C (as computed in line 7) is never empty, so the strategy always selects a successor for which the prophecy holds. This already implies that the play is winning for the $\exists$-player: Indeed, if any state $\hat{q}$ in $\mathcal{A}^\phi$ were bad, we would get $\mathfrak{P}_{\hat{q},s} = \emptyset$ for all states s, and so the set C computed in line 7 would be empty as well (as we assumed that the prophecy variables are set correctly). A detailed proof can be found in the full version [30].

C. On the Number Of Prophecies

As established in Theorem 5, the size of a complete set of prophecies is upper bounded by $|S| \cdot |Q^\phi|$. We can restrict the number of prophecies further (which is relevant in practice but does not offer an asymptotic improvement). Two states s_1, s_2 are trace equivalent, written $s_1 \equiv_{\mathit{Trace}} s_2$, if $\mathit{Traces}(\mathcal{T}_{s_1}) = \mathit{Traces}(\mathcal{T}_{s_2})$. If $s_1 \equiv_{\mathit{Trace}} s_2$, we get $\mathfrak{P}_{q,s_1} = \mathfrak{P}_{q,s_2}$ for any automaton state q, so we can restrict the prophecy construction to the equivalence class of $\equiv_{\mathit{Trace}}$.

We do not claim that our explicit prophecy construction in Section VII-A is optimal w.r.t. the number of prophecies. We can, however, show that the number of prophecies must necessarily grow with the size of the system, i.e., it cannot be constant (see the full version [30] for a proof).

Proposition 1. *There exists a $\forall\exists$ HyperLTL property φ with safety matrix and a family of transition systems $\{\mathcal{T}_n\}_{n\in\mathbb{N}}$ such that $\mathcal{T}_n$ has $\Theta(n)$-many states, and $\mathcal{T}_n \models \varphi$, and, additionally, any family of prophecies $\{\Xi_n\}_{n\in\mathbb{N}}$ where Ξ_n is complete for $\mathcal{T}_n, \varphi$ has at least size $|\Xi_n| \in \Omega(\log n)$.*

VIII. Completeness for ω-Regular Matrix

So far, the prophecy construction from Section VII is limited to the case where ϕ is a safety property. In this section, we incrementally modify the construction to support properties where ϕ expresses arbitrary ω-regular properties. To begin with, it is helpful to analyze *why* the construction from Section VII fails when moving beyond safety.

Example 6. *As a simple example to see this, we again consider the transition system in Figure 1a, generating all traces over $AP = \{a\}$. Define*

$$\varphi := \forall\pi.\exists\pi'.\Box\Diamond\left(a_{\pi'} \leftrightarrow \bigcirc a_\pi\right)$$

which expresses that π' should predict the next *step on π. Importantly, π' should not necessarily predict the next step of π at every point but at least infinitely many times. Clearly, $\mathcal{T} \models \varphi$ but $\mathfrak{V} \not\models \mathcal{G}_{\mathcal{T},\varphi}$. Let Ξ be the set of prophecies constructed in Section VII. It is easy to see that for any reachable state q (in the canonical DPA for the matrix of φ) and any trace t, it holds that $t \in \mathfrak{P}_{q,s_1}$ and $t \in \mathfrak{P}_{q,s_2}$. This is the case as choosing either state is safe, i.e., does lose the game for the $\exists$-player. Even if the current prediction is incorrect, infinitely many correct predictions are still possible in the future. The $\forall$-player can therefore set all prophecy variables to true without invalidating the premise of $\varphi^{P,\Xi}$, so $\mathfrak{V} \not\models \mathcal{G}_{\mathcal{T}^P,\varphi^{P,\Xi}}$; Ξ is an incomplete set of prophecies.*

As evident in Example 6, the root cause is that the safety prophecies provide enough information to never lose the game (i.e., encounter a situation from which the game cannot be won anymore), but this does (when moving beyond safety) not guarantee that the game is won.

A. Optimal Successors and Prophecy Construction

We begin our extension to support full ω-regularity by assuming that we can express ϕ with a *deterministic* Büchi automaton $\mathcal{A}^\phi = (Q^\phi, q_0^\phi, \delta^\phi, F^\phi)$. This is a proper extension of the safety case in Section VII (as all safety properties can be expressed with a deterministic Büchi automaton) but does not capture full ω-regularity yet (we relax this further in Section VIII-D). Note that the property in Example 6 can be recognized by a deterministic Büchi automaton. We further assume, w.l.o.g., that $\mathcal{T}$ has a unique initial state.

Following the idea from Section VII, the prophecies should explicitly tell the $\exists$-player which successor to choose. The crucial idea underlying our construction is that the prophecies should point to successor states that are safe and, additionally, satisfy that the next visit to an accepting state in F^ϕ occurs *as fast as possible* (where the speed refers to the number of steps). Always choosing such an "optimal" successor guarantees that an accepting state is visited infinitely many times. A naïve idea where prophecies express "state s is safe, and a visit to an accepting state is possible in n step" would certainly work (a strategy for the $\exists$-player would always pick a successor where the number of steps is minimal) but cannot be expressed in finitely many prophecies (the number of steps needs to be unbounded). The core idea in this section is to express optimality of a state by means of a *relative* compression with possible alternative states. Perhaps surprisingly, this is possible within the framework of ω-regular prophecies. For automaton state $q \in Q^\phi$ and system states $x, s \in S$ with $s \in \mathit{Sucs}(x)$ define $\mathfrak{P}_{q,x,s}$ as follows:

$$\Big\{ t \in \Sigma^\omega \mid \exists t' \in \mathit{Traces}(\mathcal{T}_s).\, t \otimes t' \in \mathcal{L}(\mathcal{A}^\phi_q) \land \Big[\forall s' \in \mathit{Sucs}(x). \forall t'' \in \mathit{Traces}(\mathcal{T}_{s'}).\, t \otimes t'' \in \mathcal{L}(\mathcal{A}^\phi_q) \Rightarrow \mathit{firstVisit}_{F^\phi}(\mathcal{A}^\phi_q, t \otimes t') \leq \mathit{firstVisit}_{F^\phi}(\mathcal{A}^\phi_q, t \otimes t'')\Big]\Big\}$$

Recall that $\mathit{firstVisit}_{F^\phi}(\mathcal{A}^\phi_q, t \otimes t')$ denotes the first time point that the unique run on $t \otimes t'$ visits F^ϕ. The first line in our new definition is similar to the safety case, i.e., a trace t is in $\mathfrak{P}_{q,x,s}$ if there exists a witness trace t' starting in s. In addition, we require that for any alternative successor s' of x and all traces t'' starting in s' that are also winning, the first visit to an accepting state in F^ϕ occurs at least as fast on t' as on the alternative trace t''. For a given trace $t \in \mathfrak{P}_{q,x,s}$, choosing s as the successor of x is thus optimal, in the sense that from no other successor of x there is a witness trace that visits an accepting state (strictly) sooner.

Example 7. *We revisit Example 6. With our new construction, a trace t satisfies $t \in \mathfrak{P}_{q,x,s_1}$ (for any automaton state q and any $x \in \{s_1, s_2\}$) if and only if $a \in t(1)$. That is, choosing s_1 as a successor is optimal iff this correctly predicts the next state of the $\forall$-player, i.e., a holds in the next step on t. In particular, correctly predicting the move of the $\forall$-player in the current step is better (measured in the number of steps to an accepting state) than misspredicting it now but predicting it correctly sometime in the future. A strategy that follows the recommendations of the new prophecies will* always *(instead of only infinitely many times) correctly predict the next step of the $\forall$-player and is therefore winning.*

B. On ω-Regularity

It is not immediate that $\mathfrak{P}_{q,x,s}$ is ω-regular. We begin by showing the following.

Proposition 2. *For any $q \in Q^\phi$ and $s \in \mathit{Sucs}(x)$, $\mathfrak{P}_{q,x,s}$ is ω-regular.*

Proof. We show that we can express $\mathfrak{P}_{q,x,s}$ as a QPTL formula which already gives the desired ω-regularity by Theorem 1. We make heavy use of propositional quantification in QPTL to encode paths in $\mathcal{T}$ and corresponding runs of $\mathcal{A}^\phi$. Let $V_\mathcal{T} := \{p_s \mid s \in S\}$ and $V_{\mathcal{A}^\phi} := \{p_q \mid q \in Q^\phi\}$ be propositional QPTL variables. We define formula *valid* as

$$\square \Big[un(V_\mathcal{T}) \land un(V_{\mathcal{A}^\phi}) \land \Big(\bigvee_{\substack{s \xrightarrow{\mathcal{T}} s' \\ \delta^\phi(q,(\sigma,L(s)))=q'}} p_s \land \bigcirc p_{s'} \land p_q \land \bigcirc p_{q'} \land \sigma \Big) \Big]$$

where $un(A) := \bigvee_{a \in A} \big(a \land \bigwedge_{a \neq a' \in A} \neg a'\big)$ asserts that exactly one proposition from A holds. For $\sigma \in \Sigma = 2^{AP}$ we write σ for the formula $\bigwedge_{a \in \sigma} a \land \bigwedge_{a \notin \sigma} \neg a$. This formula asserts that the propositions in $V_\mathcal{T}$ describe a valid path in $\mathcal{T}$ and the propositions in $V_{\mathcal{A}^\phi}$ a valid run of $\mathcal{A}^\phi$ where the first component in $\mathcal{A}^\phi$ is read as input and the second component is the label of the path described by $V_\mathcal{T}$.

Similarly, we use propositions $\hat{V}_\mathcal{T} := \{\hat{p}_s \mid s \in S\}$ and $\hat{V}_{\mathcal{A}^\phi} := \{\hat{p}_q \mid q \in Q^\phi\}$ to encode a second path and automaton run (as needed in the definition of $\mathfrak{P}_{q,x,s}$). We define $\widehat{valid}$ analogously to *valid* but use $\hat{p}_s$ instead of p_s and $\hat{p}_q$ instead of p_q. Now consider the following QPTL formula $\phi_{q,x,s}$:

$$\widetilde{\exists} V_\mathcal{T} \cup V_{\mathcal{A}^\phi}. \Big[p_s \land p_q \land valid \land \square \diamondsuit \bigvee_{q \in F^\phi} p_q \Big] \land \tag{4}$$

$$\Big(\widetilde{\forall} \hat{V}_\mathcal{T} \cup \hat{V}_{\mathcal{A}^\phi}. \Big[\Big(\bigvee_{s' \in Sucs(x)} \hat{p}_{s'} \Big) \land \hat{p}_q \land \widehat{valid} \land \square \diamondsuit \bigvee_{q \in F^\phi} \hat{p}_q \Big] \tag{5}$$

$$\to \Big(\bigvee_{q \notin F^\phi} \hat{p}_q \Big) \mathcal{U} \Big(\bigvee_{q \in F^\phi} p_q \Big) \Big) \tag{6}$$

This formula closely follows the definition of $\mathfrak{P}_{q,x,s}$. We existentially quantify over a path of $\mathcal{T}$ starting in s and an accompanying run of $\mathcal{A}^\phi$ (starting in q). Taking only (4) would result in a direct QPTL formula encoding of the prophecy $\mathfrak{P}_{q,s}$ from Section VII. To encode the optimality, in (5) we quantify over an alternative run that starts in some $s' \in \mathit{Sucs}(x)$ and is also accepting. Finally, (6) states that the alternative run (described via the $\hat{p}$ propositions) does not visit an accepting state as long as the existentially quantified run has not visited an accepting state. It is easy to see that the QPTL formula $\phi_{q,x,s}$ expresses $\mathfrak{P}_{q,x,s}$. □

C. Correctness Proof

We show that the resulting set of prophecies is complete. Let $\xi_{q,x,s}$ be a QPTL formula for $\mathfrak{P}_{q,x,s}$ (which exists by Proposition 2).

Theorem 6. *Assume $\mathcal{T} \models \varphi$. Define $\Xi = \{\xi_{q,x,s} \mid q \in Q^\phi, s \in \mathit{Sucs}(x)\}$ and let $P = \{p_{q,x,s} \mid q \in Q^\phi, s \in \mathit{Sucs}(x)\}$ be a fresh set of atomic propositions. Then $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Xi}}$.*

Proof sketch. To construct a winning strategy for $\mathfrak{V}$ we use a similar construction as in Section VII-B. Whenever the $\exists$-player is in a state x and q is the current state of $\mathcal{A}^\phi$ (reached on the prefix of the game), the strategy checks if any prophecy variable $p_{q,x,s}$ is set for some $s \in \mathit{Sucs}(x)$ and, if this is the case, selects any such s. Arguing the correctness of the resulting strategy is more challenging than in the safety case. We only sketch the proof here. We can assume, that the prophecies are set correctly (so the premise of $\varphi^{P,\Xi}$ holds). Under this assumption, we show that there always exists at least one successor state for which the prophecy holds. We employ a ranking argument to prove that the resulting play visits F^ϕ infinity many times. We define a function that maps each $q \in Q^\phi$, $x \in S$, and trace t to an element in $\mathbb{N} \cup \{\infty\}$ as the shortest number of steps any trace starting in a successor of x needs to take to reach an accepting state. Formally

$$opt(q, x, t) := \min_{t' \in T(q,x,t)} \mathit{firstVisit}_{F^\phi}(\mathcal{A}^\phi_q, t \otimes t')$$

where

$$T(q, x, t) := \{t' \mid \exists s \in \mathit{Sucs}(x).\, t' \in \mathit{Traces}(\mathcal{T}_s) \land \\ t \otimes t' \in \mathcal{L}(\mathcal{A}^\phi_q)\}.$$

We can establish that *opt* serves as a ranking function w.r.t. our prophecies as follows. If $q \notin F^\phi$, $s \in \mathit{Sucs}(x)$ and $t \in \mathfrak{P}_{q,x,s}$, then $opt(q', s, t[1, \infty]) < opt(q, x, t)$ (where $q' = \delta^\phi(q, (t(0), L(s))))$). That is, if a prophecy holds for $s \in \mathit{Sucs}(x)$, then the ranking function is finite and by moving to s the function either decreases *strictly* or an accepting state in F^ϕ visited. As $\mathbb{N}$ is well-founded, this implies that a visit to an accepting state occurs infinity many times. We give a detailed proof in the full version [30]. □

D. Completeness Beyond Deterministic Büchi Automata

Up to this point, we assumed that $\mathcal{A}$ is given as a deterministic Büchi automaton. We now sketch how to relax this further. For this, we assume that ϕ is given as a deterministic Rabin automaton (DRA). In a Rabin automaton, the acceptance condition is given as a set of pairs $(B_1, F_1), \ldots, (B_m, F_m)$. A run r of the automaton is accepting if there exists a $1 \leq i \leq m$

such that r visits states in B_i only finitely many times and states in F_i infinitely many times. As every parity condition is also a Rabin condition, we can translate every LTL formula ϕ (or, more generally, any ω-regular property) into an equivalent deterministic Rabin automaton.

1) One-pair Rabin Automata: To begin with, we consider the case where ϕ can be recognized by a DRA with a *single* pair, i.e., the acceptance condition consists of a set of states F that should be visited infinitely many times and a set of states B that should be visited only finitely many times. The previous construction of $\mathfrak{P}_{q,x,s}$ for deterministic Büchi automaton is incomplete as it guarantees that F is visited infinitely many times but does not ensure that B is only visited finitely many times. We sketch how the definition of $\mathfrak{P}_{q,x,s}$ is modified to support single-pair DRA and refer the reader to the full version [30] for a formal definition. We modify $\mathfrak{P}_{q,x,s}$ such that a trace t satisfies $t \in \mathfrak{P}_{q,x,s}$ if either of the following holds:

- There exists a trace t' starting in s that is winning (i.e., $t \otimes t' \in \mathcal{L}(\mathcal{A}_q^\phi)$), where the unique run *never* visits a state in B, and for all other states $s' \in \mathit{Sucs}(x)$ and any winning trace t'' starting in s' that also never visits B, $t \otimes t'$ visits a state in F at least as fast as $t \otimes t''$, or
- There does *not* exist a winning trace that never visits a state in B from any successor of x but there is a trace t' from s that is winning (but visits B at least once), and, for all states $s' \in \mathit{Sucs}(x)$ and any winning trace t'' starting in s', the *last* visit to a state in B on $t \otimes t'$ is at least as fast as the last visit on $t \otimes t''$.

Suppose the $\exists$-player follows the recommendation given by the resulting prophecies (in the sense outlined in the proof sketch of Theorem 6). By doing so, it will construct a witness trace that visits B for the last time as soon as possible and afterward (repeatedly) visits states in F as soon as possible and is therefore winning.

2) Beyond One-pair Rabin Automata: To move from a one-pair DRA to an arbitrary DRA, we simply annotate prophecies with a Rabin pair index. Given a DRA with pairs $(B_1, F_1), \ldots, (B_m, F_m)$ we compute the prophecies for a single-pair DRA for each such pair (i.e., the DRA obtained by replacing the set of Rabin pairs with a singleton set). For $q \in Q^\phi$, $s \in \mathit{Sucs}(x)$ and $1 \leq i \leq m$ we define $\mathfrak{P}_{q,x,s,i}$ as the prophecy $\mathfrak{P}_{q,x,s}$ computed on the single-pair Rabin automaton with pair (B_i, F_i) (as in Section VIII-D1). The $\exists$-player can then initially commit to one Rabin pair, say i, and afterward, always follow the recommendations of the prophecies where the index matches i (in the sense outlined in the proof sketch of Theorem 6). This strategy constructs a witness trace that is already winning for the DRA fixed to the single pair (B_i, F_i) and therefore also for the general automaton. This concludes the proof of Theorem 4.

E. On the Number of Prophecies

In our construction, the number of prophecies is linear in the size of the automaton but quadratic in the size of the system. More precisely, as we consider prophecies $\mathfrak{P}_{q,x,s}$ where $s \in \mathit{Sucs}(x)$, the number (in the size of the system) is of order $\mathcal{O}(d \cdot |S|)$ where $d = \max_{s \in S} |\mathit{Sucs}(s)|$ (which is $\mathcal{O}(|S|^2)$ in the worst case). Using a more efficient binary encoding, we can achieve an exponential decrease in the number of prophecies to $\mathcal{O}(|S| \log |S|)$ (in the size of the system). See the full version [30] for the optimized construction.

```
Input: $\mathcal{T} = (S, S_0, \varrho, L)$, $\varphi = \forall\pi.\exists\pi'.\phi$
construct DPA $\mathcal{A}^\phi = (Q^\phi, q_0^\phi, \delta^\phi, c^\phi)$
for $i = 0 \ldots |Q^\phi| \cdot |S|$ do
    for $X$ in $2_i^{Q^\phi \times S}$ do
        $\Theta \leftarrow \{\mathcal{A}_{\mathfrak{P}_{q,s}} \mid (q, s) \in X\}$
        if $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Theta}}$ then return ✓
return ✗
```

Alg. 1: Prophecy-based verification for $\forall\exists$ HyperLTL with safety matrix. The algorithm returns ✓ if $\mathcal{T} \models \varphi$ and ✗ if $\mathcal{T} \not\models \varphi$. We write $2_i^{S \times Q^\phi}$ for all subsets of $S \times Q^\phi$ with cardinality i. Automaton $\mathcal{A}_{\mathfrak{P}_{q,s}}$ represents the prophecy $\mathfrak{P}_{q,s}$. This automaton can be computed by constructing the product of $\mathcal{A}^\phi$ and $\mathcal{T}$ and is thus linear in the size of $\mathcal{T}$.

Proposition 3. *Let $\mathcal{T}$ be a (finite-state) transition system with state-space S and let φ be a $\forall^*\exists^*$ HyperLTL property such that $\mathcal{T} \models \varphi$. There exists a complete set of prophecies Ξ with $|\Xi| \in \mathcal{O}(|S| \log |S|)$.*

IX. Prophecy-based Verification and Implementation

A. Prophecy-based Verification

As the completeness result in this paper is constructive and computable, we directly obtain an algorithmic solution to the HyperLTL model checking problem. We sketch a possible algorithm for the safety case (cf. Section VII) in Algorithm 1. For each number of prophecies i (ranging from 0 to $|Q^\phi| \cdot |S|$), we consider all possible sets of prophecies X of size i, compute an automaton representation $\mathcal{A}_{\mathfrak{P}_{q,s}}$ of $\mathfrak{P}_{q,s}$ for each $(q, s) \in X$, and check if $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Theta}}$ holds. By completeness, the prophecies set of size $i = |Q^\phi| \cdot |S|$ is complete; the algorithm constitutes a sound-and-complete model checking procedure for $\forall\exists$ properties with safety matrix.[7]

We briefly discuss how we can check if $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Theta}}$. We first observe that we can write $\phi^{P,\Xi}$ (the matrix of $\varphi^{P,\Xi}$) as

$$\Big[\square\Big(\bigwedge_{j=1}^{n}(p_{j_{\pi_1}} \leftrightarrow \xi_j)\Big) \to \phi\Big] \equiv \Big[\Big(\bigvee_{j=1}^{n} \diamond(p_{j_{\pi_1}} \not\leftrightarrow \xi_j)\Big) \vee \phi\Big].$$

In Algorithm 1 we compute an NBA representation $\mathcal{A} \in \Theta$ for each prophecy. We can thus construct an NBA for $\phi^{P,\Theta}$ that is at most exponential in the size of the automata in

[7] Of course, computing the set of all prophecies identified in Theorem 5 directly (i.e., immediately setting $i = |Q^\phi| \cdot |S|$) also constitutes a complete model checker. Incrementally increasing the size of the prophecy set (as done in Algorithm 1) often results in successful verification with fewer prophecies and, in consequence, also in faster computation. If, on the other hand, the goal is to *disprove* a property, computing the full set of prophecies directly is, obviously, more efficient.

Θ, convert to a DPA, and solve the parity game $\mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Theta}}$.[8] Alternatively, we can make use of the disjunctive structure of $\phi^{P,\Xi}$ by constructing a DPA for each formula $\Diamond(p_{i_{\pi_1}} \not\leftrightarrow \xi_i)$ *individually* and then solve a generalized parity game (a game where the winning condition is a disjunction of parity objectives) [32].

Remark 5. *Algorithm 1 uses the prophecy construction for HyperLTL formulas with a safety matrix. Analogously, we could obtain a complete algorithm for arbitrary $\forall^*\exists^*$ properties by using the more general prophecy construction in Section VIII. However, generating automata representations of the prophecies is more challenging (cf. Proposition 2).*

B. Implementation and Evaluation

We have implemented Algorithm 1 (supporting $\forall^*\exists^*$ properties instead of only $\forall\exists$ properties) in a prototype model checker called `HyPro` (short for **Hy**perproperty Verification with **Pro**phecies). The novelty of `HyPro` is twofold: First, it is the first tool to fully automatically synthesize winning strategies for the $\exists$-player (based on the parity-game-based encoding). And second, `HyPro` is the first *complete* verification tool for $\forall^*\exists^*$ properties with an LTL safety matrix.

If desired by the user, `HyPro` applies a bisimulation-based preprocessing of the system.[9] We have disabled this preprocessing for our experiments.

1) Model Checking without Prophecies: We begin by evaluating `HyPro` on instances that do not require any prophecies, i.e., instances where $\mathcal{G}_{\mathcal{T},\varphi}$ is won by $\mathfrak{V}$ and so Algorithm 1 already terminates for $i = 0$. Our benchmarks consist of information-flow policies in the form of GNI and symmetry constraints (i.e., properties that require that for every trace, there exists one with the opposite outcome) on boolean programs (including those from [33]) with varying bitwidths.

We give the verification results in Table I. Our results confirm that our direct parity-game-based encoding can successfully synthesize strategies for the $\exists$-player in systems of medium size.[10] If we enable `HyPro`'s bisimulation-based preprocessing, we can verify properties of significantly larger size, as, with increasing bitwidths, the bisimulation quotient stays small. With preprocessing enabled, `HyPro` can successfully verify systems with up to 55k states within a few seconds.

TABLE I: Evaluation on instances where no prophecies are necessary to verify a property. We give the problem instance, the bitwidth of the variables (Bitwidth), the size of the program's state space after compiling to a transition system (Size), the verification outcome (Res) (✓ indicates that the property holds, ✗ that it is violated), and the overall time taken by `HyPro` (t). Times are given in seconds.

Instance	**Bitwidth**	**Size**	**Res**	t
P1 (GNI)	1-bit	17	✓	0.1
	4-bit	129		25.3
P2 (GNI)	1-bit	55	✓	0.4
P3 (GNI)	1-bit	20	✓	0.2
	3-bit	80		5.1
P4 (GNI)	1-bit	29	✓	0.2
	3-bit	113		9.2
FlipOutput (Sym)	7-bit	512	✓	9.6
FlipConjunction (Sym)	2-bit	80	✓	1.3
Switch (Sym)	3-bit	144	✓	4.4

TABLE II: Evaluation on instances where prophecies are needed to verify a property and instances where a property does not hold. We give the size of the system (Size), the number of prophecies identified using (an optimized version of) Theorem 5 (#P), the minimal cardinality of a complete prophecy set (MinP), the (cumulative) size of the automata used to represent this (minimal) complete prophecy set (SizeP), the construction time of the prophecies (t_P), the verification outcome (Res), and the overall time taken by `HyPro` (t). Times are given in seconds.

Instance	**Size**	**#P**	**MinP**	**SizeP**	t_P	**Res**	t
Predict1Small	4	10	1	4	0.1	✓	0.3
Predict1Large	20	42	1	4	0.1	✓	1.2
Predict2	4	20	3	12	1.0	✓	4.7
Example II-A	2	6	1	4	0.2	✓	0.6
Example II-B	7	14	1	6	0.1	✓	0.5
EnforceManyProph	4	16	3	12	0.8	✓	14.5
Example 3	4	20	1	3	0.5	✓	0.8
Example 4	2	10	1	2	0.1	✓	0.3
PredictLiveness	4	20	1	2	0.3	✓	0.6
MissingShift	4	5	-	-	0.2	✗	0.3
ViolationSimple	4	9	-	-	0.4	✗	0.8

We can contrast `HyPro` with the approach implemented in `MCHyper` [6], [7]. `MCHyper` requires an explicit *user-provided* strategy for the $\exists$-player, which reduces hyperproperty verification to the verification of a trace property. Obviously, strategy synthesis (as done by `HyPro`) operators on a different scale than strategy verification (as done by `MCHyper`). This motivates the coexistence of both tools: A user can either favor a fully automatic verification using `HyPro` or provide an explicit strategy and make use of the industrial-strength offered `MCHyper`. In the former, the

[8] In particular, we get that Algorithm 1 solves the model checking problem in 2-`EXPTIME` in the size of the system. We emphasize that the goal of our completeness proof is not to derive an efficient model checking algorithm. As we will see in Section IX-B, the actual number of prophecies needed is often much smaller and research into more efficient prophecy constructions is an interesting direction for future work (cf. Section X-B).

[9] For two bisimilar systems $\mathcal{T}_1$ and $\mathcal{T}_2$ (see, e.g., [21] for a formal definition) it holds that $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}_1,\varphi}$ iff $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}_2,\varphi}$ for every φ. Therefore, we can apply strategy-based verification to the (in many cases much smaller) bisimulation quotient. Note $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}_1,\varphi}$ and $\mathfrak{V} \vdash \mathcal{G}_{\mathcal{T}_1,\varphi}$ are, in general, not equivalent when $\mathcal{T}_1$ and $\mathcal{T}_2$ are only trace equivalent.

[10] Note that the size column in Table I gives the size of an individual system. If we, e.g., verify GNI, the size of the resulting parity game is cubic in the size of the system (as GNI involves three trace quantifiers).

tedious, error-prone, and time-consuming task of writing an explicit strategy by hand is avoided, whereas the latter supports larger systems.

2) Model Checking with Prophecies: As a second benchmark, we compiled a collection of very small transition systems that cannot be verified without the use of prophecies. Our benchmarks include programs where non-deterministic choices need to be resolved before the information needed is provided or where predictions on future behavior are demanded. The results are given in Table II. None of the existing solvers [6], [7], [33] can verify any of these instances. Moreover, based on our completeness result, HyPro is the first tool that can prove that a property does *not* hold.

Even though the prophecies computed by HyPro are only guaranteed to be complete for properties with safety matrix, the construction empirically also works for properties beyond safety (such as Example 3). For Example 3, HyPro computes the prophecy depicted as an NBA in Figure 5, which precisely captures the information needed by the ∃-player, i.e., it determines if b_π never holds at an even position. Note again that this prophecy is not LTL-definable.

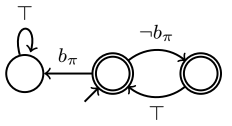


Fig. 5: Prophecy automaton constructed by HyPro for Example 3.

In Table II, we observe that the actual number of prophecies needed to verify a property (MinP) is often much smaller than the overall number of prophecies (#P). This observation is encouraging, as it indicates that the information needed by the ∃-player is concise, i.e., expressible with few automata.

We remark that the direct prophecy construction in Algorithm 1 (and implemented in HyPro) is, obviously, limited to very small systems as the number of prophecies scales linearly in the size of the system (also see Section X-B).

X. Discussion

A. Further Applications of Prophecy-based Verification

The primary motivation for our work is rooted in the need for efficient and accurate (meaning complete) verification methods for hyperproperties with quantifier alternation. Nevertheless, prophecies for hyperproperty verification are also useful beyond just constituting a complete proof method. We highlight two such cases in the context of explainable verification results and hyperproperty verification on software.

1) Prophecies for Explainable Verification: Ideally, a verification tool should not only verify that a property holds but convince the user (of, e.g., a security-critical library) *why* this is the case [34]. Certifying verification results of safety trace properties (or k-safety hyperproperties) are well understood as the verification tool can provide an (inductive) invariant on the system. Understanding verification outcomes in the presence of quantifier alteration, such as for GNI, is much more challenging. Prophecy-based verification naturally provides a user-understandable certificate. If a property is verified, a user is provided with (1) an explicit strategy σ for the ∃-player, (2) an invariant on the plays produced by σ, and (3) a finite set of prophecies needed by σ. This triple allows for a deep investigation into the system as the prophecies directly indicate which future decisions are relevant. The user can even interactively step through the strategy and prophecies and explore the trace constructed by the strategy.

2) Verification of Infinite-state Systems: Prophecies are also useful in the context of hyperproperty verification on infinite-state systems. For such systems, complementation-based verification is, unsurprisingly, impossible. In contrast, strategy-based verification is applicable (see, e.g., [13]). Prophecies can strengthen the ∃-player and result in more successful verification instances.

B. Future Work

While HyPro demonstrates that an explicit prophecy construction is applicable in practice, verification is, obviously, restricted to very small systems. In fact, the direct prophecy-based construction implemented in HyPro is, most likely, easily outperform by complementation-based verification approaches (which are currently not implemented in any tool). This leaves the construction of more efficient methods to synthesize relevant prophecies as a particularly interesting direction for future work. A natural idea would be to, instead of using a *fixed* prophecy construction (as in Algorithm 1), focus on counter-example guided approaches that iteratively add prophecies by analyzing a spoiling strategy for the ∀-player in $\mathcal{G}_{\mathcal{T}^P, \varphi^{P,\Xi}}$. Existing techniques for LTL learning [35], [36], or automaton learning [37]–[39] can be used to identify prophecies that distinguish traces on which different future behavior by the ∃-player is necessary. This would exhibit much of the benefits of prophecy-based verification (in particular, the explainability of verification results) while scaling well in the size of the system. As we establish with this paper, a well-chosen prophecy generation (that, in the limit, enumerates enough prophecies) would constitute a complete verification method. Moreover, as demonstrated in Table II, the actual number of prophecies needed in practice is often small.

XI. Conclusion

In this paper, we have provided a formal footing for the use of prophecy variables for hyperproperty verification by giving a precise characterization of their expressive power. The main result is that prophecies turn strategy-based verification into a complete verification method for arbitrary $\forall^*\exists^*$ properties. Our completeness proof is informative in the sense that it provides an explicit, effective, and finite-state-representable (ω-regular) construction of the prophecies. This new foundation asks for further research to devise prophecy-based (complete) verification methods that scale to larger systems.

Acknowledgments

This work was partially supported by the German Research Foundation (DFG) in project 389792660 (*Foundations of Perspicuous Software Systems*, TRR 248). R. Beutner carried out this work as a member of the Saarbrücken Graduate School of Computer Science.

References

[1] M. R. Clarkson and F. B. Schneider, "Hyperproperties," in *IEEE Computer Security Foundations Symposium, CSF 2008*. IEEE Computer Society, 2008. [Online]. Available: https://doi.org/10.1109/CSF.2008.7

[2] M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez, "Temporal logics for hyperproperties," in *International Conference on Principles of Security and Trust, POST 2014*, ser. Lecture Notes in Computer Science, vol. 8414. Springer, 2014. [Online]. Available: https://doi.org/10.1007/978-3-642-54792-8_15

[3] A. W. Roscoe, J. Woodcock, and L. Wulf, "Non-interference through determinism," *J. Comput. Secur.*, vol. 4, no. 1, 1996. [Online]. Available: https://doi.org/10.3233/JCS-1996-4103

[4] D. McCullough, "Noninterference and the composability of security properties," in *IEEE Symposium on Security and Privacy, SP 1988*. IEEE Computer Society, 1988. [Online]. Available: https://doi.org/10.1109/SECPRI.1988.8110

[5] G. Barthe, P. R. D'Argenio, and T. Rezk, "Secure information flow by self-composition," *Math. Struct. Comput. Sci.*, vol. 21, no. 6, 2011. [Online]. Available: https://doi.org/10.1017/S0960129511000193

[6] B. Finkbeiner, M. N. Rabe, and C. Sánchez, "Algorithms for model checking HyperLTL and HyperCTL*," in *International Conference on Computer Aided Verification, CAV 2015*, ser. Lecture Notes in Computer Science, vol. 9206. Springer, 2015. [Online]. Available: https://doi.org/10.1007/978-3-319-21690-4_3

[7] N. Coenen, B. Finkbeiner, C. Sánchez, and L. Tentrup, "Verifying hyperliveness," in *International Conference on Computer Aided Verification, CAV 2019*, ser. Lecture Notes in Computer Science, vol. 11561. Springer, 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-25540-4_7

[8] M. Abadi and L. Lamport, "The existence of refinement mappings," *Theor. Comput. Sci.*, vol. 82, no. 2, 1991. [Online]. Available: https://doi.org/10.1016/0304-3975(91)90224-P

[9] G. Barthe, J. M. Crespo, and C. Kunz, "Beyond 2-safety: Asymmetric product programs for relational program verification," in *International Symposium on Logical Foundations of Computer Science, LFCS 2013*, ser. Lecture Notes in Computer Science, vol. 7734. Springer, 2013. [Online]. Available: https://doi.org/10.1007/978-3-642-35722-0_3

[10] H. Unno, T. Terauchi, and E. Koskinen, "Constraint-based relational verification," in *International Conference on Computer Aided Verification, CAV 2021*, ser. Lecture Notes in Computer Science, vol. 12759. Springer, 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-81685-8_35

[11] L. Lamport and F. B. Schneider, "Verifying hyperproperties with TLA," in *IEEE Computer Security Foundations Symposium, CSF 2021*. IEEE, 2021. [Online]. Available: https://doi.org/10.1109/CSF51468.2021.00012

[12] T. Hsu, C. Sánchez, and B. Bonakdarpour, "Bounded model checking for hyperproperties," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2021*, ser. Lecture Notes in Computer Science, vol. 12651. Springer, 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-72016-2_6

[13] R. Beutner and B. Finkbeiner, "Software verification of hyperproperties beyond k-safety," in *International Conference on Computer Aided Verification, CAV 2022*, ser. Lecture Notes in Computer Science. Springer, 2022.

[14] B. Cook, H. Khlaaf, and N. Piterman, "On automation of CTL* verification for infinite-state systems," in *International Conference on Computer Aided Verification, CAV 2015*, ser. Lecture Notes in Computer Science, vol. 9206. Springer, 2015. [Online]. Available: https://doi.org/10.1007/978-3-319-21690-4_2

[15] N. A. Lynch and F. W. Vaandrager, "Forward and backward simulations: I. untimed systems," *Inf. Comput.*, vol. 121, no. 2, 1995. [Online]. Available: https://doi.org/10.1006/inco.1995.1134

[16] R. Jung, R. Lepigre, G. Parthasarathy, M. Rapoport, A. Timany, D. Dreyer, and B. Jacobs, "The future is ours: prophecy variables in separation logic," *Proc. ACM Program. Lang.*, vol. 4, no. POPL, 2020. [Online]. Available: https://doi.org/10.1145/3371113

[17] Z. Zhang, X. Feng, M. Fu, Z. Shao, and Y. Li, "A structural approach to prophecy variables," in *Annual Conference on Theory and Applications of Models of Computation, TAMC 2012*, ser. Lecture Notes in Computer Science, vol. 7287. Springer, 2012. [Online]. Available: https://doi.org/10.1007/978-3-642-29952-0_12

[18] V. Vafeiadis, "Modular fine-grained concurrency verification," Ph.D. dissertation, University of Cambridge, UK, 2008.

[19] O. Padon, J. Hoenicke, K. L. McMillan, A. Podelski, M. Sagiv, and S. Shoham, "Temporal prophecy for proving temporal properties of infinite-state systems," *Formal Methods Syst. Des.*, vol. 57, no. 2, 2021. [Online]. Available: https://doi.org/10.1007/s10703-021-00377-1

[20] B. Cook and E. Koskinen, "Making prophecies with decision predicates," in *ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2011*. ACM, 2011. [Online]. Available: https://doi.org/10.1145/1926385.1926431

[21] C. Baier and J. Katoen, *Principles of model checking*. MIT Press, 2008.

[22] S. Safra, "On the complexity of omega-automata," in *Annual Symposium on Foundations of Computer Science, FOCS 1988*. IEEE Computer Society, 1988. [Online]. Available: https://doi.org/10.1109/SFCS.1988.21948

[23] N. Piterman, "From nondeterministic büchi and streett automata to deterministic parity automata," *Log. Methods Comput. Sci.*, vol. 3, no. 3, 2007. [Online]. Available: https://doi.org/10.2168/LMCS-3(3:5)2007

[24] B. Alpern and F. B. Schneider, "Defining liveness," *Inf. Process. Lett.*, vol. 21, no. 4, 1985. [Online]. Available: https://doi.org/10.1016/0020-0190(85)90056-0

[25] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," in *International Conference on Computer Aided Verification, CAV 1999*, ser. Lecture Notes in Computer Science, vol. 1633. Springer, 1999. [Online]. Available: https://doi.org/10.1007/3-540-48683-6_17

[26] D. A. Martin, "Borel determinacy," *Annals of Mathematics*, vol. 102, no. 2, 1975.

[27] V. Diekert and P. Gastin, "First-order definable languages," in *Logic and Automata: History and Perspectives*, ser. Texts in Logic and Games, vol. 2. Amsterdam University Press, 2008.

[28] A. P. Sistla, *Theoretical issues in the design and verification of distributed systems*. Harvard University, 1983.

[29] M. Y. Vardi and P. Wolper, "Reasoning about infinite computations," *Inf. Comput.*, vol. 115, no. 1, 1994. [Online]. Available: https://doi.org/10.1006/inco.1994.1092

[30] R. Beutner and B. Finkbeiner, "Prophecy variables for hyperproperty verification," *CoRR*, vol. abs/2206.01797, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2206.01797

[31] R. Beutner, D. Carral, B. Finkbeiner, J. Hofmann, and M. Krötzsch, "Deciding hyperproperties combined with functional specifications," in *Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2022*. ACM, 2022. [Online]. Available: https://doi.org/10.1145/3531130.3533369

[32] K. Chatterjee, T. A. Henzinger, and N. Piterman, "Generalized parity games," in *International Conference on Foundations of Software Science and Computational Structures, FOSSACS 2007*, ser. Lecture Notes in Computer Science, vol. 4423. Springer, 2007. [Online]. Available: https://doi.org/10.1007/978-3-540-71389-0_12

[33] R. Beutner and B. Finkbeiner, "A temporal logic for strategic hyperproperties," in *International Conference on Concurrency Theory, CONCUR 2021*, ser. LIPIcs, vol. 203. Dagstuhl, 2021. [Online]. Available: https://doi.org/10.4230/LIPIcs.CONCUR.2021.24

[34] H. Chockler, J. Y. Halpern, and O. Kupferman, "What causes a system to satisfy a specification?" *ACM Trans. Comput. Log.*, vol. 9, no. 3, 2008. [Online]. Available: https://doi.org/10.1145/1352582.1352588

[35] D. Neider and I. Gavran, "Learning linear temporal properties," in *Formal Methods in Computer Aided Design, FMCAD 2018*, N. Bjørner and A. Gurfinkel, Eds. IEEE, 2018. [Online]. Available: https://doi.org/10.23919/FMCAD.2018.8603016

[36] C. Lemieux, D. Park, and I. Beschastnikh, "General LTL specification mining (T)," in *IEEE/ACM International Conference on Automated Software Engineering, ASE 2015*. IEEE Computer Society, 2015. [Online]. Available: https://doi.org/10.1109/ASE.2015.71

[37] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, 1987. [Online]. Available: https://doi.org/10.1016/0890-5401(87)90052-6

[38] B. Finkbeiner, L. Haas, and H. Torfah, "Canonical representations of k-safety hyperproperties," in *IEEE Computer Security Foundations Symposium, CSF 2019*. IEEE, 2019. [Online]. Available: https://doi.org/10.1109/CSF.2019.00009

[39] A. Farzan, Y. Chen, E. M. Clarke, Y. Tsay, and B. Wang, "Extending automated compositional verification to the full class of omega-regular languages," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2008*, vol. 4963. Springer, 2008. [Online]. Available: https://doi.org/10.1007/978-3-540-78800-3_2