Controller Synthesis for Hyperproperties

Borzoo Bonakdarpour
Department of Computer Science and Engineering
Michigan State University, USA
Email: borzoo@msu.edu

Bernd Finkbeiner
Reactive Systems Group
Saarland University, Germany
Email: finkbeiner@cs.uni-saarland.de

***Abstract*—We investigate the problem of *controller synthesis* for *hyperproperties* specified in the temporal logic HyperLTL. Hyperproperties are system properties that relate multiple execution traces. Hyperproperties can elegantly express information-flow policies like noninterference and observational determinism. The controller synthesis problem is to automatically design a controller for a *plant* that ensures satisfaction of a given specification in the presence of the environment or adversarial actions. We show that the controller synthesis problem is decidable for HyperLTL specifications and finite-state plants. We provide a rigorous complexity analysis for different fragments of HyperLTL and different system types: tree-shaped, acyclic, and general graphs.**

I. Introduction

In *program synthesis*, an algorithm automatically constructs a program that satisfies a given high-level specification given in some formal logic [1], [2]. The *controller synthesis* [3] problem is a specific form of program synthesis where we ask whether the *controllable transitions* of a given system, called the *plant*, can be selected in such a way that the resulting restricted system satisfies the given specification. Algorithms for this problem automate the design of a controller that ensures the satisfaction of the specification even in the presence of an *adversary* modeled by *uncontrollable transitions*. Synthesis guarantees correctness by construction and it enables users to refrain from the error-prone process of developing software and, instead, to focus on only analyzing the functional behavior of the system. Thus, program synthesis exhibits its power particularly in automating the generation of intricate and complex parts of a system.

A particular area where synthesis can play an important role is the construction of systems where the flow of information is critical, for example to ensure *confidentiality*. This is because even a short transient violation of *security* or *privacy* policies may result in leaking private or highly sensitive information, compromising safety, or interrupting vital public or social services. The hacking of the emergency system in the city of Dallas [4], the *Heartbleed* error that leaked records of 4.5 million patients [5], the data leak at Yahoo, which resulted in stealing 500 million accounts [6], and the *Goto Fail* bug, where the encryption of more than 300 million devices was broken [7], only scratch the surface of high-profile examples of such security breaches. Synthesis with respect to security policies is of particular interest as it can construct protocols that are guaranteed to behave correctly in the presence of adversarial attacks or untrusted parties. Also, given a set of actions and primitives of parties that participate in a protocol, synthesis can be used to synthesize trusted third parties that mediate between other parties (see Section III for a concrete example of an application of controller synthesis to non-repudiation protocols).

In order to express and reason about information-flow security policies, we use the powerful formalism of *hyperproperties* [8]. Hyperproperties elevate trace properties from a set of execution traces to sets of sets of execution traces. *Temporal logics for hyperproperties* such as HyperLTL [9] have been introduced to give clear syntax and semantics to hyperproperties. HyperLTL allows for the simultaneous quantification over the temporal behavior of *multiple* execution traces. Atomic propositions are indexed to refer to specific traces. For example, *noninterference* [10] between a secret input h and a public output o can be specified in HyperLTL by stating that, for all pairs of traces π and π', if the input is the same for all input variables I except h, then the output o must be the same at all times:

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

Another prominent example is *generalized noninterference* (GNI) [11], which can be expressed as the following HyperLTL formula:

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

The existential quantifier is needed to allow for nondeterminism. Generalized noninterference permits nondeterminism in the low-observable behavior, but stipulates that low-security outputs may not be altered by the injection of high-security inputs.

Techniques for automatically constructing systems that satisfy a given set of information-flow properties are still in their infancy. The general synthesis problem for HyperLTL is known to be undecidable as soon as the formula contains two universal quantifiers [12]. To remedy this problem, *bounded synthesis* [12], [13] restricts the search to implementations up to a given bound on the number of states. Bounded synthesis is decidable, but still very difficult, because the transition structure of the controller must be found. Bounded synthesis has been successfully applied to examples such as the dining cryptographers [14], but does not seem to scale to large systems. Another prominent approach is *program repair* [15], where, for a given program that does not satisfy a HyperLTL property, transitions are eliminated until the program satisfies the property. Program repair has the advantage that the repair directly works on the state space of the given program. However, program repair is not reactive in the sense that it does not allow for modeling the actions of an adversarial

DOI 10.1109/CSF49147.2020.00033

HyperLTL fragment	Tree	Acyclic	General
E^*	L-complete *(Theorem 1)*	NL-complete *(Theorem 5)*	NL-complete *(Theorem 9)*
E^*A			PSPACE-complete *(Theorem 11)*
AE^*	P-complete *(Theorem 2)*	Σ_2^p-complete *(Theorem 8)*	
AA^+	NP-complete *(Corollary 1)*	NP-complete *(Theorem 6)*	NP-complete *(Theorem 10)*
$(E^*A^*)^k$, $k \geq 2$		Σ_k^p-complete *(Theorem 8)*	$(k-1)$-EXPSPACE-complete *(Theorem 11)*
$(A^*E^*)^k$, $k \geq 1$		Σ_{k+1}^p-complete *(Theorem 8)*	
$(A^*E^*)^*$		PSPACE *(Corollary 3)*	NONELEMENTARY *(Corollary 4)*

TABLE I: Complexity of the HyperLTL controller synthesis problem in the size of the plant, where k is the number of quantifier alternations in the formula.

environment, which is vital to dealing with information-flow security.

This limitation is addressed by *controller synthesis*. Controller synthesis is based on a *plant*, which describes the system behavior in terms of controllable and uncontrollable transitions. Like program repair, controller synthesis has the advantage that the state space is already given. Unlike program repair, controller synthesis distinguishes between controllable and uncontrollable transitions and therefore considers an adversary. The synthesized controller guarantees that, no matter how the adversary behaves, the specification is satisfied.

In this paper, we study the controller synthesis problem of finite-state systems with respect to HyperLTL specifications. We provide a detailed analysis of the complexity of the controller synthesis problem for different fragments of HyperLTL and different shapes of the plants motivated by the following observations:

- The security and privacy policies of interest vary in the quantifier structure that is needed to express the policy in HyperLTL. Typical examples are $\forall\forall$ (noninterference), $\forall\forall\exists$ (generalized noninterference), and $\forall\exists$ (noninference) [9]. Data minimization, a popular privacy technique, is of the form $\forall\forall\exists\exists$ [16]. The non-repudiation protocol discussed in Section III is specified with a HyperLTL formula of the form $\exists\forall\forall\forall$.
- The protocols and systems of interest vary in the shape of their state graphs. We are interested in *general*, *acyclic*, and *tree-shaped* plants. The need for investigating the controller synthesis problem for tree-shaped and acyclic plants stems from two reasons. First, many trace logs are in the form of a simple linear collection of the traces seen so far. Or, for space efficiency, the traces are organized by common prefixes and assembled into a tree-shaped graphs, or by common prefixes as well as suffixes assembled into an acyclic graphs. These example scenarios can be used to synthesize protocols, as has been done for synthesizing distributed algorithms [17] that respect certain safety and liveness constraints. In the context of security/privacy, the example scenarios would be used to synthesize a complete protocol that satisfy a set of hyperproperties. The second reason is that, tree-shaped and acyclic graphs often occur as the natural representation of the state space of a protocol. For example, certain security protocols, such as non-repudiation, authentication, and session-based protocols (e.g., TLS, SSL, SIP) go through a finite sequence of *phases*, resulting in an acyclic plant A detailed example of synthesizing a tree-shaped non-repudiation protocol is presented in Section III.

We investigate the difficulty of the controller synthesis problem with respect to different combinations of the quantifier structure of the HyperLTL formula and the shape of the plant.

Table I summarizes the results of this paper. The complexities are in the size of the plant. This *system complexity* is the most relevant complexity in practice, because the system tends to be much larger than the specification. Our results show that the shape of the plant plays a crucial role in the complexity of the controller synthesis problem.

- **Trees.** For trees, the complexity in the size of the plant does not go beyond NP. The problem for the existential alternation-free fragment and the fragment with one quantifier alternation where the leading quantifier is existential is L-complete. The problem for the fragment with one quantifier alternation where the leading quantifier is universal is P-complete. However, the problem becomes NP-complete as soon as there are two leading universal quantifiers.
- **Acyclic graphs.** For acyclic plants, the complexity is NL-complete for the alternation-free fragment and the fragment with one quantifier alternation, where the leading quantifier is existential. Similar to tree-shaped graphs, the problem becomes NP-complete as soon as there are two leading universal quantifiers. Furthermore, the complexity is in the level of the polynomial hierarchy that corresponds to the number

of quantifier alternations.

- **General graphs.** For general plants, the complexity is NL-complete for the existential fragment and NP-complete for the universal fragment. The complexity is PSPACE-complete for the fragment with one quantifier alternation and $(k-1)$-EXPSPACE-complete in the number k of quantifier alternations.

Surprisingly, the complexities identified in Table I are very much aligned with those reported for the program repair problem [15]. The main exceptions are the complexity for the universal alternation-free fragment in tree-shaped and acyclic graphs that are NP-complete in the case of controller synthesis and are L-complete and NL-complete, respectively, in case of the repair problem. This fragment is of particular interest, as it hosts many of the important information-flow security policies.

We believe that the results of this paper provide the fundamental understanding of the controller synthesis problem for secure information flow and pave the way for further research on developing efficient and scalable techniques.

Organization: The remainder of this paper is organized as follows. In Section II, we review HyperLTL. We present a detailed motivating example in Section III. The formal statement of the controller synthesis problem is in Section IV. Section V presents our results on the complexity of controller synthesis for HyperLTL in the size of tree-shaped plants. Sections VI and VII present the results on the complexity of synthesis in acyclic and general graphs, respectively. We discuss related work in Section VIII and conclude with a discussion of future work in Section IX.

II. Preliminaries

A. Plants

Let AP be a finite set of *atomic propositions* and $\Sigma = 2^{\mathsf{AP}}$ be the *alphabet*. A *letter* is an element of Σ. A *trace* $t \in \Sigma^\omega$ over alphabet Σ is an infinite sequence of letters: $t = t(0)t(1)t(2)\ldots$

Definition 1: A *plant* is a tuple $\mathcal{P} = \langle S, s_{init}, \mathfrak{c}, \mathfrak{u}, L\rangle$, where

- S is a finite set of *states*;
- $s_{init} \in S$ is the *initial state*;
- $\mathfrak{c}, \mathfrak{u} \subseteq S \times S$ are respectively sets of of *controllable* and *uncontrollable transitions*, where $\mathfrak{c} \cap \mathfrak{u} = \{\}$, and
- $L : S \to \Sigma$ is a *labeling function* on the states of $\mathcal{P}$.

We require that for each $s \in S$, there exists $s' \in S$, such that $(s, s') \in \mathfrak{c} \cup \mathfrak{u}$.

Figure 1 shows an example plant, where transition (s_{init}, s_1) is an uncontrollable transition while the rest are controllable, and $L(s_{init}) = \{a\}, L(s_3) = \{b\}$, etc. The *size* of the plant is the number of its states. The directed graph $\mathcal{F} = \langle S, \mathfrak{c} \cup \mathfrak{u}\rangle$ is called the *frame* of the plant $\mathcal{P}$. A *loop* in $\mathcal{F}$ is a finite sequence $s_0 s_1 \cdots s_n$, such that $(s_i, s_{i+1}) \in \mathfrak{c} \cup \mathfrak{u}$, for all $0 \leq i < n$, and $(s_n, s_0) \in \mathfrak{c} \cup \mathfrak{u}$. We call a frame *acyclic*, if the only loops are self-loops on terminal states, i.e., on states that have no other outgoing transition.

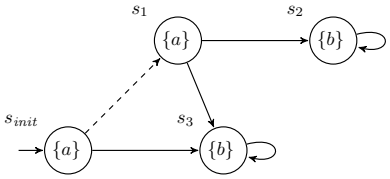

Fig. 1: An acyclic plant.

We call a frame *tree-shaped*, or, in short, a *tree*, if every state s has a unique state s' with $(s', s) \in \mathfrak{c} \cup \mathfrak{u}$, except for the root node, which has no predecessor, and the leaf nodes, which, again because of Definition 1, additionally have a self-loop but no other outgoing transitions. In some cases, the system at hand is given as a tree-shaped or acyclic plant. Examples include session-based security protocols and space-efficient execution logs, because trees allow us to organize the traces according to common prefixes and acyclic graphs according to both common prefixes and common suffixes.

A *path* of a plant is an infinite sequence of states $s(0)s(1)\cdots \in S^\omega$, such that:

- $s(0) = s_{init}$, and
- $(s(i), s(i+1)) \in \mathfrak{c} \cup \mathfrak{u}$, for all $i \geq 0$.

A trace of a plant is a trace $t(0)t(1)t(2)\cdots \in \Sigma^\omega$, such that there exists a path $s(0)s(1)\cdots \in S^\omega$ with $t(i) = L(s(i))$ for all $i \geq 0$. We denote by $\mathsf{Traces}(\mathcal{P})$ the set of all traces of $\mathcal{P}$ with paths that start in state s_{init}.

B. The Temporal Logic HyperLTL

HyperLTL [9] is an extension of linear-time temporal logic (LTL) for hyperproperties. The syntax of HyperLTL formulas is defined inductively by the following grammar:

$$\varphi ::= \exists \pi.\varphi \mid \forall \pi.\varphi \mid \phi$$
$$\phi ::= \mathsf{true} \mid a_\pi \mid \neg\phi \mid \phi \vee \phi \mid \phi \ \mathcal{U}\ \phi \mid \bigcirc \phi$$

where $a \in \mathsf{AP}$ is an atomic proposition and π is a trace variable from an infinite supply of variables $\mathcal{V}$. The Boolean connectives $\neg$ and $\vee$ have the usual meaning, $\mathcal{U}$ is the temporal *until* operator and $\bigcirc$ is the temporal *next* operator. We also consider the usual derived Boolean connectives, such as $\wedge$, $\Rightarrow$, and $\Leftrightarrow$, and the derived temporal operators *eventually* $\Diamond\varphi \equiv \mathsf{true}\ \mathcal{U}\ \varphi$ and *globally* $\Box\varphi \equiv \neg\Diamond\neg\varphi$. The quantified formulas $\exists\pi$ and $\forall\pi$ are read as 'along some trace π' and 'along all traces π', respectively. For example, the following formula:

$$\forall\pi.\forall\pi'.\Box(a_\pi \Leftrightarrow a_{\pi'})$$

intends to express that every pair of traces should always agree on the position of proposition a.

The semantics of HyperLTL is defined with respect to a trace assignment, a partial mapping $\Pi\colon \mathcal{V} \to \Sigma^\omega$. The assignment with empty domain is denoted by $\Pi_\emptyset$. Given a trace assignment Π, a trace variable π, and a concrete trace $t \in \Sigma^\omega$, we denote by $\Pi[\pi \to t]$ the assignment that coincides with Π everywhere but at π, which is mapped to trace t. Furthermore, $\Pi[j, \infty]$ denotes the assignment mapping each trace π in Π's

domain to $\Pi(\pi)(j)\Pi(\pi)(j{+}1)\Pi(\pi)(j{+}2)\cdots$. The satisfaction of a HyperLTL formula φ over a trace assignment Π and a set of traces $T \subseteq \Sigma^\omega$, denoted by $T, \Pi \models \varphi$, is defined as follows:

$$
\begin{array}{lll}
T, \Pi \models \mathsf{true} & & \\
T, \Pi \models a_\pi & \text{iff} & a \in \Pi(\pi)(0), \\
T, \Pi \models \neg\psi & \text{iff} & T, \Pi \not\models \psi, \\
T, \Pi \models \psi_1 \vee \psi_2 & \text{iff} & T, \Pi \models \psi_1 \text{ or } T, \Pi \models \psi_2, \\
T, \Pi \models \bigcirc \psi & \text{iff} & T, \Pi[1, \infty] \models \psi, \\
T, \Pi \models \psi_1 \,\mathcal{U}\, \psi_2 & \text{iff} & \exists i \geq 0 : T, \Pi[i, \infty] \models \psi_2 \wedge \\
 & & \forall j \in [0, i) : T, \Pi[j, \infty] \models \psi_1, \\
T, \Pi \models \exists \pi.\ \psi & \text{iff} & \exists t \in T : T, \Pi[\pi \to t] \models \psi, \\
T, \Pi \models \forall \pi.\ \psi & \text{iff} & \forall t \in T : T, \Pi[\pi \to t] \models \psi.
\end{array}
$$

We say that a set T of traces satisfies a sentence φ, denoted by $T \models \phi$, if $T, \Pi_\emptyset \models \varphi$. If the set $T = \mathsf{Traces}(\mathcal{P})$ is generated by a plant $\mathcal{P}$, we write $\mathcal{P} \models \varphi$.

III. Motivating Example

In order to motivate the decision problem described in Section IV, we consider the application of controller synthesis to construct a *trusted third party* for a *fair non-repudiation protocol*. The purpose of a non-repudiation protocol is to allow two parties to exchange a message without any party being able to deny having participated in the exchange. For this purpose, the recipient of the message obtains a *non-repudiation of origin* (NRO) evidence and the sender of the message obtains a *non-repudiation of receipt* (NRR) evidence. The protocol is *effective* if it is possible to successfully transmit the message to the recipient and the evidence to both parties. The protocol is *fair* if it is furthermore *impossible* for one party to obtain the evidence without the other party *also* receiving the evidence. Fairness in an effective protocol cannot be obtained without an external trusted agent, called the *trusted third party*, which mediates the message exchange and ensures the delivery of the evidence. We now consider the problem of synthesizing such a trusted third party from the specification of fairness and effectiveness.

Let A be the sender of the message, B be the receiver, and T the trusted third party. A has 5 possible actions, corresponding to sending the message m or the NRO to either B or to T, or, alternatively, doing nothing at all:

$$
\begin{aligned}
Act_A = \{&A{\to}B{:}m, A{\to}T{:}m, \\
&A{\to}B{:}NRO, A{\to}T{:}NRO, A{:}skip\}.
\end{aligned}
$$

Likewise, B can send the NRR to A or T, or do nothing at all:

$$Act_B = \{B{\to}A{:}NRR, B{\to}T{:}NRR, B{:}skip\}.$$

T can send the NRR to A, the NRO or m to B, or do nothing at all:

$$Act_T = \{T{\to}A{:}NRR, T{\to}B{:}NRO, T{\to}B{:}m, T{:}skip\}.$$

We'll assume that the three parties take turns and interact in a fixed number of rounds. The plant, thus, is *tree-shaped*, where along each branch the states belong first to A, then to T, then to B, and then again A, then T, then B, etc. States that belong to A branch according to the actions in Act_A, likewise states that belong to T branch according to Act_T, and states that belong to B according to Act_B. We label the states with the atomic propositions

$$\mathsf{AP} = \{m, NRR, NRO\} \cup Act_A \cup Act_T \cup Act_B,$$

where m indicates that B has received the message, NRO that B has received the NRO, NRR that A has received the NRR, and one of the actions in $Act_A \cup Act_T \cup Act_B$ that the respective action has just occurred. Since we are interested in synthesizing the trusted third party, the outgoing transitions of states belonging to T are controllable, all other transitions are uncontrollable. That is, we are only allowed to manipulate the behavior of the trusted third party and not the original participants in the protocol.

We specify *effectiveness* by requiring that there is a sequence of actions π such that the message, the NRR, and the NRO get received. For *fairness*, we additionally require that if either A executes the actions according to π (and B behaves arbitrarily) or B executes π (and A behaves arbitrarily), then it must still hold that the NRR gets received if and only if the NRO gets received.

$$
\begin{aligned}
&\varphi = \exists\pi.\forall\pi'.\ (\diamondsuit m_\pi) \wedge (\diamondsuit NRR_\pi) \wedge (\diamondsuit NRO_\pi) && \text{(effectiveness)} \\
&\wedge \Big((\square \textstyle\bigwedge_{a \in Act_A} a_\pi \Leftrightarrow a_{\pi'}) \Rightarrow \big((\diamondsuit NRR_{\pi'}) \Leftrightarrow (\diamondsuit NRO_{\pi'})\big)\Big) && \text{(fairness for } A\text{)} \\
&\wedge \Big((\square \textstyle\bigwedge_{a \in Act_B} a_\pi \Leftrightarrow a_{\pi'}) \Rightarrow \big((\diamondsuit NRR_{\pi'}) \Leftrightarrow (\diamondsuit NRO_{\pi'})\big)\Big) && \text{(fairness for } B\text{)}
\end{aligned}
$$

The following trusted third party is a correct solution to the controller synthesis problem:

T_{correct} :
(1) skip until $A{:}m{\to}T$;
(2) skip until $A{:}NRO{\to}T$;
(3) $T{\to}B{:}m$;
(4) skip until $B{\to}T{:}NRR$;
(5) $T{\to}B{:}NRO$;
(6) $T{\to}A{:}NRR$.

An *incorrect* solution to the controller synthesis problem would, for example, be a trusted third party that does not wait for the NRR from B before forwarding the NRO from A to B: now B could quit the protocol without ever providing the NRR.

$T_{\text{incorrect}}$:
(1) skip until $A{:}m{\to}T$;
(2) skip until $A{:}NRO{\to}T$;
(3) $T{\to}B{:}m$;
(4) $T{\to}B{:}NRO$;
(5) skip until $B{\to}T{:}NRR$;
(6) $T{\to}A{:}NRR$.

$T_{\text{incorrect}}$ violates the fairness requirement for A. Note, however, that φ also admits the following, somewhat counter-intuitive, solution T_{strange}:

T_{strange} :
(1) skip until $A{:}m{\to}B$;
(2) $T{\to}B{:}NRO$;
(3) $T{\to}A{:}NRR$.

In this solution, T transmits the NRO and NRR even though the message m was sent directly from A to B *without* ever

going through T. The reason, why T_{strange} can do this, is that it can choose its actions based on complete information, i.e., based on the position in the tree. If we wish to restrict the possible solutions for T to only those that are based only on the messages actually received by T, we need to add a consistency condition for incomplete information:

$$\forall\pi.\forall\pi'.\ \Big(\square \bigwedge_{o\in Obs_T} o_\pi \leftrightarrow o_{\pi'}\Big) \Rightarrow \Big(\square \bigwedge_{a\in Act_T} a_\pi \Leftrightarrow a_{\pi'}\Big)$$

where

$$Obs_T = \{A{\to}T{:}m, A{\to}T{:}NRO, B{\to}T{:}NRR\}$$

consists of the actions that send a message to T.

Constructing a trusted third party for fair non-repudiation thus requires solving a controller synthesis problem for a tree-shaped plant. For effectiveness and fairness, a HyperLTL formula with quantifier prefix $\exists\pi.\forall\pi'$ is required, for consistency under incomplete information, additionally a HyperLTL formula with quantifier prefix $\forall\pi.\forall\pi'$.

IV. Problem Statement

The *controller synthesis problem* is the following decision problem. Let $\mathcal{P} = \langle S, s_{init}, \mathfrak{c}, \mathfrak{u}, L\rangle$ be a plant and φ be a closed HyperLTL formula, where $\mathcal{P}$ may or may not satisfy φ. Does there exist a plant $\mathcal{P}' = \langle S', s'_{init}, \mathfrak{c}', \mathfrak{u}', L'\rangle$ such that:

- $S' = S$,
- $s'_{init} = s_{init}$,
- $\mathfrak{c}' \subseteq \mathfrak{c}$,
- $\mathfrak{u}' = \mathfrak{u}$,
- $L' = L$, and
- $\mathcal{P}' \models \varphi$?

In other words, the goal of the controller synthesis problem is to identify a plant $\mathcal{P}'$, whose set of traces is a subset of the traces of $\mathcal{P}$ that satisfies φ, by only restricting the controllable transitions and without removing any uncontrollable transitions. Note that since the witness to the decision problem is a plant, following Definition 1, it is implicitly implied that in $\mathcal{P}'$, for every state $s \in S'$, there exists a state s' such that $(s, s') \in \mathfrak{c}' \cup \mathfrak{u}'$. I.e., the synthesis does not create a *deadlock* state.

We use the following notation to distinguish the different variations of the problem:

CS[Fragment, Frame Type],

where

- CS is the *controller synthesis* decision problem as described above;
- Fragment is one of the following for φ:
  - We use regular expressions to denote the order and pattern of repetition of quantifiers. For example, $\mathsf{E}^*\mathsf{A}^*$-HyperLTL denotes the fragment, where an arbitrary (possibly zero) number of existential quantifiers is followed by an arbitrary (possibly zero) number of universal quantifiers. Also, AE^+-HyperLTL means a lead universal quantifier followed by one or more existential quantifiers. $\mathsf{E}^{\leq 1}\mathsf{A}^*$-HyperLTL denotes the fragment, where zero or one existential quantifier is followed by an arbitrary number of universal quantifiers.
  - (EA)k-HyperLTL, for $k \geq 0$, denotes the fragment with k alternations and a lead existential quantifier, where $k = 0$ means an alternation-free formula with only existential quantifiers;
  - (AE)k-HyperLTL, for $k \geq 0$, denotes the fragment with k alternations and a lead universal quantifier, where $k = 0$ means an alternation-free formula with only universal quantifiers,
  - HyperLTL is the full logic HyperLTL, and
- Frame Type is either tree, acyclic, or general.

V. Complexity of Controller Synthesis for Tree-shaped Graphs

We begin by analyzing the complexity of the controller synthesis problem for tree-shaped plants.

A. The $\mathsf{E}^*\mathsf{A}$ Fragment

Our first result is that the controller synthesis problem for tree-shaped plants can be solved in logarithmic time in the size of the plant for the fragment with only one quantifier alternation, where the leading quantifier is existential and there is only one universal quantifier. This fragment is the least expensive to deal with in tree-shaped plants and, interestingly, the complexity is the same as for the model checking [18] and model repair problems [15].

Theorem 1: CS[E^*A-HyperLTL, tree] is L-complete in the size of the plant.

Proof: We first show membership to L. We note that the number of traces in a tree is bounded by the number of states, i.e., the size of the plant. The synthesis algorithm enumerates (using the complete plant) all possible assignments for the existential trace quantifiers. For each existential trace variable, we need a counter up to the number of traces, which requires only a logarithmic number of bits in the size of the plant.

For the universal quantifier, the algorithm first checks if assigning one of the traces that already have been assigned to the existential quantifiers now to the universal quantifier will violate the formula. If so, the existential assignment is disregarded. Otherwise, the algorithm proceeds to check if the traces that must implicitly be present (because of uncontrollable transitions or because of potential deadlocks) satisfy the formula. For this purpose, the algorithm evaluates the nodes of the tree bottom-up, such that a controller exits iff the root node evaluates positively.

Let n be a node in the tree whose children are all leaves. If all outgoing transitions are controllable, then n evaluates positively iff the assigning the trace of one the children to the universally quantified variable satisfies the formula. If at least one of the transitions is uncontrollable, then n evaluates positively iff all such assignments satisfy the formula.

Now, let m be a node further upward in the tree. If all outgoing transitions are controllable, then the node evaluates to true iff some child evaluates to true. We evaluate bottom-up the first child by moving to the first leaf node reachable from the first child. If the evaluation is negative, we move to the first leaf node reachable from the second child etc. If the evaluation of one of the children is positive we proceed to m's parent with a positive evaluation. If we have reached the last child with a negative evaluation, we proceed to the parent with a negative evaluation.

If there is at least one uncontrollable transition, we disregard the controllable transitions and only step through the uncontrollable transitions, initially by moving to the first leaf reachable through the first uncontrollable transition. If the evaluation is positive, we move to the first leave node reachable from the second child etc. If the evaluation of one of the children is negative we proceed to the parent with a negative evaluation. If we have reached the last child with a positive evaluation, we proceed to m's parent with a positive evaluation.

The algorithm terminates when the root node has been evaluated. During the entire bottom-up traversal, we only need to store a pointer to a single node of the tree in memory, which can be done with logarithmically many bits.

In order to show completeness, we prove that the controller synthesis problem for the existential fragment is L-hard. The L-hardness for CS[E*-HyperLTL, tree] follows from the L-hardness of ORD [19]. ORD is the graph-reachability problem for directed line graphs. Graph reachability from s to t can be checked with the synthesis problems for $\exists\pi.\ \Diamond(s_\pi \wedge \Diamond t_\pi)$ or $\forall\pi.\ \Diamond(s_\pi \wedge \Diamond t_\pi)$. ■

B. The AE* Fragment

We now study the complexity of the controller synthesis problem for the fragment with only one quantifier alternation, where the leading quantifier is universal.

Theorem 2: CS[AE*-HyperLTL, tree] is P-complete in the size of the plant.

Proof: We show membership to P with a simple marking algorithm. Let $\varphi = \forall\pi_1.\exists\pi_2.\ \psi$. We begin by marking all leaves. We then proceed in several rounds, such that in each round, at least one mark is removed. We, hence, terminate within linearly many rounds in the size of the tree-shaped plant.

In each round, we go through all marked leaves v_1 and instantiate π_1 with the trace leading to v_1. We then again go through all marked leaves v_2 and instantiate π_2 with the trace leading to v_2, and check ψ on the pair of traces, which can be done in linear time [18]. If the check is successful for some instantiation of π_2, we leave v_1 marked, otherwise we remove the mark. If no mark was removed by the end of the round, we terminate. Each round of the marking algorithm takes linear time in the size of the tree, the complete algorithm thus takes quadratic time. Once the marking algorithm has terminated, we remove all branches of the tree that are not marked. As established by the final round of the marking algorithm, the remaining tree satisfies φ. For additional existential quantifiers, we go in each round through the possible instantiations of all the existential quantifiers, which can be done in polynomial time. In case we reach a situation, where the only way to satisfy the formula is to remove an uncontrollable transition, then the answer to the synthesis problem is negative.

For the lower bound, we reduce HORN-SAT, which is P-complete, to the synthesis problem for AE* formulas. HORN-SAT is the following problem:

> Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of propositional variables. A *Horn clause* is a clause over X with at most one positive literal. Is $y = y_1 \wedge y_2 \wedge \cdots \wedge y_m$, where each y_j is a Horn clause for all $j \in [1, m]$, satisfiable? That is, does there exist an assignment of truth values to the variables in X, such that all clauses of y evaluate to true?

In the following, we work with a modified version of HORN-SAT, where every clause consists of two negative and one positive literals. In order to transform any arbitrary Horn formula y to another one y' that consists of two negative and one positive literals, we apply the following:

- To ensure that every clause contains a positive literal, we introduce a fresh variable $\bot$ with the intended meaning "false". We add $\bot$ as a positive literal to all clauses that have no positive literal.
- To ensure that every clause contains at least two negative literals, we introduce a fresh variable $\top$ with the intended meaning "true". We add $\top$ as a negative literal to all clauses that have no negative literals. (Clauses with only one negative literal count as clauses with two negative literals with two identical negative literals.)
- To ensure that no clause contains more than two negative literals, we reduce the number of negative literals as follows: Let l_1 and l_2 be two negative literals in a clause with more than two negative literals. We introduce a fresh variable f and replace l_1 and l_2 with $\neg f$; we furthermore add $\{l_1, l_2, f\}$ as a new clause.
- In order to account for the intended meaning of $\top$ and $\bot$, we modify the HORN-SAT problem to check if there exists a truth assignment to the variables in $X \cup \{\top, \bot\}$ (union fresh variables f to break clauses as described above), such that all clauses in y evaluate to true *and* $\bot$ evaluates to false and $\top$ evaluates to true.

For example, we transform Horn formula:

$$y = (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_1)$$

to the following:

$$\begin{aligned} y' =& (\neg x_1 \vee \neg x_2 \vee f) \wedge \\ & (\neg x_3 \vee \neg f \vee x_4) \wedge (\neg x_2 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_1 \vee \bot) \end{aligned}$$

Hence, the set of propositional variables for the transformed formula y' is updated to $X = \{\bot, x_1, x_2, x_3, x_4, f, \top\}$. Since the modified problem and the original problem are obviously equivalent, it follows that the modified problem is P-complete as well. We now describe our mapping (see Fig. 2 for an example).

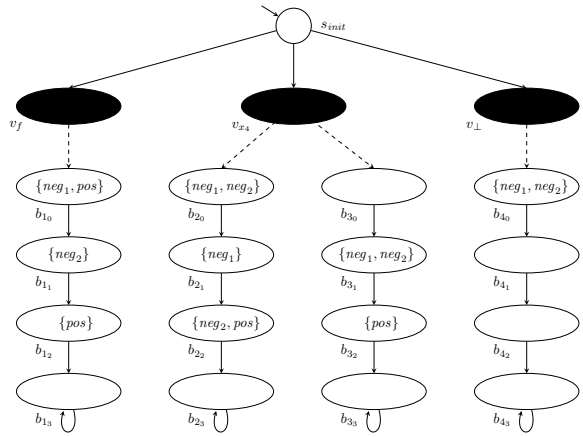

Fig. 2: The plant for Horn formula $y = (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_1)$.

Plant. We translate the (modified) HORN-SAT problem to a tree-shaped plant $\mathcal{P} = \langle S, s_{init}, \mathfrak{c}, \mathfrak{u}, L \rangle$ as follows. Our general idea is to create a tree, where each branch represents a clause in the input HORN-SAT problem and states of each branch are labeled according to the bitstring encoding of the literals in that clause. Thus, the length of each branch needs to be $\log(|X|)$. We now present the details:

- *(Atomic propositions* AP*)* We include atomic propositions neg_1 and neg_2 to indicate negative literals and pos for the positive literals in a Horn clause. Thus,
$$\mathsf{AP} = \{neg_1, neg_2, pos\}.$$
- *(Set of states* S*)* We now identify the members of S:
  - First, we include an initial state s_{init}, which is labeled with the empty set of atomic propositions.
  - For each propositional variable $x \in X$ that appears as a positive literal in a clause, we add a state v_x. The idea here is to ensure that if a positive literal appears on some clause in the synthesized plant, then all clauses with the same positive literal must be preserved during synthesis. These state are not labeled by any atomic propositions in AP.
  - For each clause y_j, where $j \in [1, m]$, we include a bitstring that represents which literals participate in y_j. That is, we include the following in S:
$$\Big\{ b_{j_i} \mid j \in [1, m] \wedge i \in [0, \log(|X|)) \Big\}.$$
We represent literals in each clause by labeling the states of the clause according to the appearance of the propositional variables $x_i \in X$ (i.e., the updated set including $\top$, $\perp$, and fresh f variables), where $i \in [0, \log(|X|))$, as a negative or positive literal. More specifically, let $y_j = \{\neg x_{n_1} \vee \neg x_{n_2} \vee x_p\}$ be a Horn clause. We label states $b_{j_0}, b_{j_1}, \ldots, b_{j_{\log(X)-1}}$ by atomic proposition neg_1 according to the bitsequence of n_1, atomic proposition neg_2 according to the bitsequence of n_2, and atomic proposition pos according to the bitsequence of p. We reserve values 0 and $|X| - 1$ for $\perp$ and $\top$, respectively (see Fig 2).
- *(Uncontrollable transitions* $\mathfrak{u}$*)* For each state v_x, we add an uncontrollable transition from v_x to any state b_{j_0}, where propositional variable x appears as a positive literal in clause y_j. These transitions ensure that if a variable that appears a positive literal in two or more clauses, all or none of the associated clauses are preserved.
- *(Controllable transitions* $\mathfrak{c}$*)* We represent each Horn clause as a branch of the plant. That is, we include the following transitions:
  - We connect the states that represent bitstrings as follows:
$$\Big\{ (b_{j_i}, b_{j_{i+1}}) \mid j \in [1, m] \wedge i \in [0, \log(X)) \Big\}.$$
  - We also connect the initial state to each v_x state, where x is a propositional variable that appears as a positive literal in some Horn clause.
  - Finally, we add a self-loop to the end of each branch, that is,
$$\Big\{ (b_{j_{\log(X)-1}}, b_{j_{\log(X)-1}}) \mid j \in [1, m] \Big\}.$$

It is easy to see that the plant that represents the Horn clauses is a tree. It branches into the paths that represent the clauses (see Fig. 2).

HyperLTL formula. We interpret the synthesized plant as a solution to HORN-SAT assigning false to every variable x that appears as a positive literal on some path but v_x is *not* reachable from the initial state and true to every variable x' that appears as a positive literal on some path but $v_{x'}$ *is* reachable from the initial state. We define a HyperLTL formula that ensures that this valuation satisfies the clause set. Let
$$\varphi_{\mathsf{map}} = \varphi_\perp \wedge \varphi_\top \wedge \varphi_C$$
be a HyperLTL formula with the following conjuncts:

- Formula $\varphi_\top$ enforces that $\top$ is assigned to true. This is expressed by requiring that, on all traces, $\top$ does not appear as a positive literal. That is,
$$\varphi_\top = \forall \pi_1. \Diamond(\neg pos_{\pi_1}).$$
- Formula $\varphi_\perp$ stipulates that $\perp$ is assigned to false. This is expressed by requiring that there exists a trace where $\perp$ appears as a positive literal. That is,
$$\varphi_\perp = \exists \pi_2. \Box(\neg pos_{\pi_2}).$$
- Formula φ_C ensures that all clauses are satisfied. This is expressed as a forall-exists formula that requires, for every trace in the synthesized plant, that for one of the variables that appear as negative literals in

the clause, there must exist a trace where the same variable appears as the positive literal. That is,

$$\varphi_C = \forall\pi_1.\exists\pi_2.\square\Big((neg_{1_{\pi_1}} \Leftrightarrow pos_{\pi_2}) \vee (neg_{2_{\pi_1}} \Leftrightarrow pos_{\pi_2})\Big).$$

Overall, φ_{map} needs only one universal and one existential quantifier and it is straightforward to see that the input HORN-SAT formula is satisfiable if and only if the answer to the controller synthesis problem is affirmative. ■

C. The Full Logic

We now turn to full HyperLTL. We show that the controller synthesis problem is in NP. NP-hardness holds already for the fragment with two universal quantifiers.

Theorem 3: CS[HyperLTL, tree] is in NP in the size of the plant.

Proof: We nondeterministically guess a solution $\mathcal{P}'$ in polynomial time. Since determining whether or not $\mathcal{P}' \models \varphi$ can be solved in logarithmic space [18], the synthesis problem is in NP. ■

Theorem 4: CS[AA-HyperLTL, tree] is NP-hard in the size of the plant.

Proof: We reduce the 3-SAT problem to the controller synthesis problem. The 3SAT problem is as follows:

> Let $\{x_1, x_2, \ldots, x_n\}$ be a set of propositional variables. Given is a Boolean formula $y = y_1 \wedge y_2 \wedge \cdots \wedge y_m$, where each y_j, for $j \in [1, m]$, is a disjunction of exactly three literals. Is y satisfiable? That is, does there exist an assignment of truth values to $x_1, x_2, \ldots, x_n$, such that y evaluates to true.

We now present a mapping from an arbitrary instance of 3SAT to the synthesis problem of a tree-shaped plant and a HyperLTL formula of the form $\forall\pi.\forall\pi'.\psi$. Then, we show that the plant satisfies the HyperLTL formula if and only if the answer to the 3SAT problem is affirmative. Figure 3 shows an example.

Plant $\mathcal{P} = \langle S, s_{init}, \mathfrak{c}, \mathfrak{u}, L\rangle$:

- *(Atomic propositions* AP*)* We include two atomic propositions: pos and neg to mark the positive and negative literals in each clause. Thus,
$$\mathsf{AP} = \{pos, neg\}.$$
- *(Set of states S)* We now identify the members of S:
  - First, we include an initial state s_{init}. Then, for each clause y_j, where $j \in [1, m]$, we include a state r_j.
  - Let $y_j = (l, l', l'')$ be a clause in the 3SAT formula. We include the following set of states:
$$\Big\{v_{j_i}, v'_{j_i}, v''_{j_i} \mid i \in [1, n]\Big\},$$
where n is the number of propositional variables. If $l = x_i$ is in y_j, then we label state v_{j_i} with proposition pos. If $l = \neg x_i$ in y_j, then we label state v_{j_i} with proposition neg. We analogously label v' and v'' associated to l' and l'', respectively.

  Thus, we have
$$S = \Big\{r_j \mid j \in [1, m]\Big\} \cup \Big\{v_{j_i}, v'_{j_i}, v''_{j_i} \mid i \in [1, n] \ \wedge \ j \in [1, m]\Big\}.$$
- *(Uncontrollable transitions* $\mathfrak{u}$*)* We include an *uncontrollable* transition (s_{init}, r_j), for each clause y_j in the 3SAT formula, where $j \in [1, m]$:
$$\mathfrak{u} = \Big\{(s_{init}, r_j) \mid j \in [1, m]\Big\}.$$
These uncontrollable transitions ensure that during synthesis, all clauses are preserved.
- *(Controllable transitions* $\mathfrak{c}$*)* We now identify the members of $\mathfrak{c}$:
  - We connect all the states corresponding to the states corresponding to the propositional variables in a sequence. That is, we include the following transitions for each $j \in [1, m]$:
$$\Big\{(r_j, v_{j_1}), (r_j, v'_{j_1}), (r_j, v''_{j_1})\Big\} \cup \Big\{(v_{j_i}, v_{j_{i+1}}), (v'_{j_i}, v'_{j_{i+1}}), (v''_{j_i}, v''_{j_{i+1}}) \mid i \in [1, n)\Big\}$$
  - We also add a self-loop at each leaf state:
$$\Big\{(v_{j_n}, v_{j_n}), (v'_{j_n}, v'_{j_n}), (v''_{j_n}, v''_{j_n}) \mid j \in [1, m]\Big\}$$

HyperLTL formula: The HyperLTL formula in our mapping is the following:

$$\varphi_{\mathsf{map}} = \forall\pi_1.\forall\pi_2.\square\Big(\neg pos_{\pi_1} \vee \neg neg_{\pi_2}\Big)$$

We now show that the given 3SAT formula is satisfiable if and only if the plant obtained by our mapping can be controlled to satisfy the HyperLTL formula φ_{map}:

- ($\Rightarrow$) Suppose the 3SAT formula y is satisfiable, i.e., there exists an assignment to the propositional variables $x_1, x_2, \ldots, x_n$ that satisfies y. This implies that each y_j becomes true, which in turn means that there exists at least one literal in each y_j that evaluates to true. Now, given this assignment, we identify a plant $\mathcal{P}'$ that satisfies the conditions of the controller synthesis problem stated in Section IV. Suppose $y_j = (l \vee l' \vee l'')$, for some $j \in [1, m]$. Also, suppose that $l = x_i$, for some $i \in [1, n]$. If $x_i = \mathsf{true}$ in the answer to the 3SAT problem, then we keep states $v_{j_1}, v_{j_2}, \ldots, v_{j_n}$ and all incoming and outgoing transitions to them. We also remove states $v'_{j_1}, v'_{j_2}, \ldots, v'_{j_n}$ and $v''_{j_1}, v''_{j_2}, \ldots, v''_{j_n}$. Likewise, suppose that $l = \neg x_i$, for some $i \in [1, n]$. If $x_i = \mathsf{false}$ in the answer to the 3SAT problem, then we keep states $v_{j_1}, v_{j_2}, \ldots, v_{j_n}$ and all incoming and outgoing transitions to them. We also remove states $v'_{j_1}, v'_{j_2}, \ldots, v'_{j_n}$ and $v''_{j_1}, v''_{j_2}, \ldots, v''_{j_n}$ and all incoming and outgoing transitions to them. The case for literals l', l'' and states v', v'' follows trivially.
  It is straightforward to see that the plant obtained after removing the aforementioned states satisfies φ_{map}.

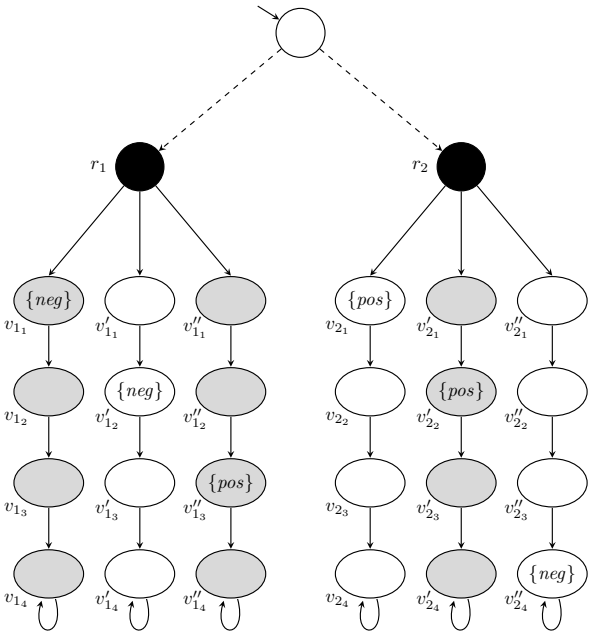

Fig. 3: The plant for the 3SAT formula $(\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_4)$. The truth assignment $x_1 = \mathsf{true}$, $x_2 = \mathsf{false}$, $x_3 = \mathsf{false}$, $x_4 = \mathsf{false}$ renders the tree with white branches, i.e., the grey branches are removed during synthesis.

This is because a propositional variable cannot be simultaneously true and false in the answer to the 3SAT problem. Thus, if we keep the states corresponding to a true variable x_i, the branches where some v_{j_i} is labeled by neg is removed. The same argument holds for states v' and v'' and the case where a variable evaluates to false.

- ($\Leftarrow$) Suppose the answer to the synthesis problem is affirmative, i.e., there is a synthesized plant $\mathcal{P}'$ that satisfies the HyperLTL formula φ_{map}. This means that (1) all the r states are preserved, since we cannot remove uncontrollable transitions, and (2) there are no pairs of v, v', or v'' states at the same height of the tree of the synthesized plant, such that both pos and neg are true. We now describe how one can obtain a truth assignment to the propositional variables that satisfies the input 3SAT formula. Suppose that state $v_{j_1}, v_{j_2}, \ldots, v_{j_n}$ appear in $\mathcal{P}'$ for some i and j, such that some state v_{j_i} is labeled by pos. We assign truth value true to variable x_i. This assignment makes clause y_j true, since x_i is a literal in y_j. On the contrary, if state v_{j_i} is labeled by neg, then we assign truth value false to variable x_i. This assignment makes clause y_j true, since $\neg x_i$ is a literal in y_j. Same argument holds for states v' and v''. Furthermore, since all r states are preserved all clauses evaluate to true and, hence, y evaluates to true.

This concludes the proof. ∎

Corollary 1: CS[AA-HyperLTL, tree] is NP-complete in the size of the plant.

Corollary 1 shows a significant difference between program repair [15] and controller synthesis: while the program repair problem for the AA^* fragment is L-complete, the problem becomes NP-complete for controller synthesis.

Corollary 2: CS[HyperLTL, tree] is NP-complete in the size of the plant.

VI. Complexity of Controller Synthesis for Acyclic Graphs

In this section, we analyze the complexity of the controller synthesis problem for acyclic plants.

A. The Alternation-free Fragment

We start with the existential fragment. It turns out that for this fragment, controller synthesis and model checking are actually the same problem; hence, the complexity of controller synthesis is NL-complete, as known from model checking [18].

Theorem 5: CS[E^*-HyperLTL, acyclic] is NL-complete in the size of the plant.

Proof: For existential formulas, the synthesis problem is equivalent to the model checking problem. A given plant satisfies the formula iff there is a solution to the synthesis problem, as we are only dealing with existential quantifiers. If the formula is satisfied, the witness to the synthesis problem is simply the original plant. Since the model checking problem for existential formulas over acyclic graphs is NL-complete [18, Theorem 2], the same holds for the synthesis problem. ∎

We now switch to the universal fragment, where the complexity of the problem jumps to NP-complete.

Theorem 6: CS[A^*-HyperLTL, acyclic] is NP-complete in the size of the plant.

Proof: For the upper bound, one can guess a solution to the synthesis problem and verify in polynomial time. The lower bound follows the lower bound of CS[AA-HyperLTL, tree] shown in Theorem 4. ∎

B. Formulas with Quantifier Alternation

We first build on Theorem 6 to study the complexity of the synthesis problem for the $\mathsf{E}^*\mathsf{A}^*$ Fragment.

Theorem 7: CS[$\mathsf{E}^*\mathsf{A}^*$-HyperLTL, acyclic] is in Σ_2^{p} in the size of the plant.

Proof: We show membership to Σ_2^{p}. Since the plant is acyclic, the length of the traces is bounded by the number of states. We can thus nondeterministically guess the witness to the existentially quantified traces in polynomial time, and then solve the problem for the remaining formula, which has only universal quantifiers. By Theorem 6 (i.e., NP-hardness of the problem for the A^* fragment), it holds that CS[$\mathsf{E}^*\mathsf{A}^*$-HyperLTL, acyclic] is in Σ_2^{p}. ∎

Next, we consider formulas where the number of quantifier alternations is bounded by a constant k. We show that changing the frame structure from trees to acyclic graphs results in a

significant increase in complexity (see Table I). The complexity of the synthesis problem is similar to the model checking problem, with the synthesis problem being one level higher in the polynomial hierarchy (cf. [18]). This also means that complexity of the problem is aligned with the complexity of the model repair problem [15].

Theorem 8: For $k \geq 2$, CS[(EA)k-HyperLTL, acyclic] is $\Sigma^{\mathsf{p}}_{\mathsf{k}}$-complete in the size of the plant. For $k \geq 1$, CS[(AE)k-HyperLTL, acyclic] is $\Sigma^{\mathsf{p}}_{\mathsf{k+1}}$-complete in the size of the plant.

Proof: We show membership in $\Sigma^{\mathsf{p}}_{\mathsf{k}}$ and $\Sigma^{\mathsf{p}}_{\mathsf{k+1}}$, respectively, as follows. Suppose that the first quantifier is existential. Since the plant is acyclic, the length of the traces is bounded by the number of states. We can thus nondeterministically guess the witness to the existentially quantified traces in polynomial time, and then verify the correctness of the guess by model checking the remaining formula, which has $k-1$ quantifier alternations and begins with a universal quantifier. The verification can be done in $\Pi^{\mathsf{p}}_{\mathsf{k-1}}$ [18, Theorem 3]. Hence, the synthesis problem is in $\Sigma^{\mathsf{p}}_{\mathsf{k}}$.

If the first quantifier is universal, we apply the same procedure except that we only guess the solution to the synthesis problem (there are no leading existential quantifiers). In this case, the formula for the model checking problem has k quantifier alternations. Hence, we solve the model checking problem in $\Pi^{\mathsf{p}}_{\mathsf{k}}$ and the synthesis problem in $\Sigma^{\mathsf{p}}_{\mathsf{k+1}}$.

We give a matching lower bound for CS[(AE)k-HyperLTL, acyclic]. Since the (AE)k-HyperLTL formulas are contained in the (EA)$k+1$-HyperLTL formulas (not using the outermost existential quantifiers), this also provides a matching lower bound for CS[(EA)k-HyperLTL, acyclic].

We establish the lower bound for CS[(AE)k-HyperLTL, acyclic] via a reduction from the *quantified Boolean formula* (QBF) satisfiability problem [20]:

> *Given is a set of Boolean variables,* $\{x_1, x_2, \ldots, x_n\}$*, and a quantified Boolean formula*
>
> $$y = \mathbb{Q}_1 x_1 . \mathbb{Q}_1 x_2 \ldots \mathbb{Q}_{n-1} x_{n-1} . \mathbb{Q}_n x_n . (y_1 \wedge y_2 \wedge \cdots \wedge y_m)$$
>
> *where each* $\mathbb{Q}_i \in \{\forall, \exists\}$ *(*$i \in [1, n]$*) and each clause* y_j *(*$j \in [1, m]$*) is a disjunction of three literals (3CNF). Is* y *true?*

If $\mathbb{Q}_1 = \exists$ and y is restricted to at most k alternations of quantifiers, then QBF satisfiability is complete for $\Sigma^{\mathsf{p}}_{\mathsf{k}}$. We note that in the given instance of the QBF problem:

- The clauses may have more than three literals, but three is sufficient of our purpose;
- The inner Boolean formula has to be in conjunctive normal form in order for our reduction to work;
- Without loss of generality, the variables in the literals of the same clause are different (this can be achieved by a simple pre-processing of the formula), and
- If the formula has k alternations, then it has $k+1$ alternation *depths*. For example, formula

$$\forall x_1 . \exists x_2 . (x_1 \vee \neg x_2)$$

has one alternation, but two alternation depths: one for $\forall x_1$ and the second for $\exists x_2$. By $d(x_i)$, we mean the alternation depth of Boolean variable x_i.

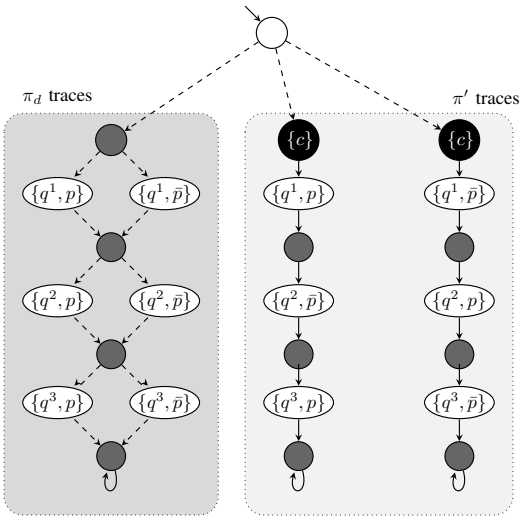

Fig. 4: Plant for the formula $y = \exists x_1 . \forall x_2 . \exists x_3 . (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$.

We now present, for $k \geq 1$, a mapping from an arbitrary instance of QBF with k alternations and where $\mathbb{Q}_1 = \exists$ to the synthesis problem of an acyclic plant and a HyperLTL formula with $k-1$ quantifier alternations and a leading universal quantifier. Then, we show that the plant can be pruned so that it satisfies the HyperLTL formula if and only if the answer to the QBF problem is affirmative.

The reduction is similar to the reduction from QBF satisfiability to the HyperLTL model checking problem [18, Theorem 3] except for the treatment of the outermost existential quantifiers. In the reduction to the model checking problem, these quantifiers are translated to trace quantifiers, resulting in a HyperLTL formula with k quantifier alternations and a leading existential quantifier. In the reduction to the synthesis problem, the outermost existential quantifiers are resolved by the pruning of controllable transitions. For this reason, it suffices to build a HyperLTL formula with one less quantifier alternation, i.e., with $k-1$ quantifier alternations, and a leading universal quantifier.

In the following, we first describe the plant from the reduction to the model checking problem [18, Theorem 3] and then describe the necessary additions for the reduction to the synthesis problem. Figure 4 shows an example.

Plant $\mathcal{P} = \langle S, s_{init}, \mathfrak{c}, \mathfrak{u}, L \rangle$**:**

- *(Atomic propositions* AP*)* For each alternation depth $d \in [1, k+1]$, we include an atomic proposition q^d. We furthermore include three atomic propositions: c is used to mark the clauses, p is used to force clauses to become true if a Boolean variable appears in a clause, and proposition $\bar{p}$ is used to force clauses to become

true if the negation of a Boolean variable appears in a clause in our reduction.

- *(Set of states S)* We now identify the members of S:
  - First, we include an initial state s_{init} and a state r_0. Then, for each clause y_j, where $j \in [1,m]$, we include a state r_j, labeled by proposition c.
  - For each clause y_j, where $j \in [1,m]$, we introduce the following $2n$ states:
    $$\Big\{v_i^j, u_i^j \mid i \in [1,n]\Big\}.$$
    Each state v_i^j is labeled with propositions $q^{d(x_i)}$, and with p if x_i is a literal in y_j, or with $\bar{p}$ if $\neg x_i$ is a literal in y_j.
  - For each Boolean variable x_i, where $i \in [1,n]$, we include three states s_i, $\bar{s}_i$, and $\hat{s}_i$. Each state s_i (respectively, $\bar{s}_i$) is labeled by p and $q^{d(x_i)}$ (respectively, $\bar{p}$ and $q^{d(x_i)}$).

  Thus,
  $$S = \big\{s_{init}\big\} \cup \big\{r_j \mid j \in [0,m]\big\} \cup \big\{v_i^j, u_i^j, s_i, \bar{s}_i, \hat{s}_i \mid i \in [1,n] \wedge j \in [1,m]\big\}.$$
- *(Uncontrollable transitions $\mathfrak{u}$)* The set of uncontrollable transitions include the following:
  - We add outgoing transitions from the initial state of $\mathcal{P}$ to clause states as well as the state that represent the first propositional variable. The idea here is to make these transitions uncontrollable to ensure that during synthesis the clauses and the diamond structure for all alternation depths except $k+1$ are preserved.
  - For each $i \in [1,n]$, we include transitions $(s_i, \hat{s}_i)$ and $(\bar{s}_i, \hat{s}_i)$. For each $i \in [1,n)$, we include transitions $(\hat{s}_i, s_{i+1})$ and $(\hat{s}_i, \bar{s}_{i+1})$.

  Thus,
  $$\begin{aligned}\mathfrak{u} = &\big\{(s_{init}, r_j) \mid j \in [0,m]\big\} \cup \\ &\big\{(r_0, s_1), (r_0, \bar{s}_1)\big\} \cup \\ &\big\{(s_i, \hat{s}_i), (\bar{s}_i, \hat{s}_i) \mid i \in [1,n]\big\} \cup \\ &\big\{(\hat{s}_i, s_{i+1}), (\hat{s}_i, \bar{s}_{i+1}) \mid i \in [1,n)\big\}.\end{aligned}$$
- *(Controllable transitions $\mathfrak{c}$)* We now identify the members of $\mathfrak{c}$:
  - We add transitions (r_j, v_1^j) for each $j \in [1,m]$.
  - For each $i \in [1,n]$ and $j \in [1,m]$, we include transitions (v_i^j, u_i^j). For each $i \in [1,n)$ and $j \in [1,m]$, we include transitions (u_i^j, v_{i+1}^j).
  - We include two transitions (r_0, s_1) and $(r_0, \bar{s}_1)$.
  - Finally, we include self-loops $(\hat{s}_n, \hat{s}_n)$ and (u_n^j, u_n^j), for each $j \in [1,m]$.

  Thus,
  $$\begin{aligned}\mathfrak{c} = &\big\{(r_j, v_1^j), (u_n^j, u_n^j) \mid j \in [1,m]\big\} \cup \\ &\big\{(v_i^j, u_i^j) \mid i \in [1,n] \wedge j \in [1,m]\big\} \cup \\ &\big\{(u_i^j, v_{i+1}^j) \mid i \in [1,n) \wedge j \in [1,m]\big\}.\end{aligned}$$

HyperLTL formula: The role of the HyperLTL formula in our reduction is to ensure that the QBF instance is satisfiable iff the HyperLTL formula is satisfied on the solution to the synthesis problem:

$$\begin{aligned}\varphi_{\mathsf{map}} = \forall\pi_{k+1}.\forall\pi_k.\exists\pi_{k-1}\cdots\exists\pi_2.\forall\pi_1.\forall\pi'.\\ \Bigg(\bigwedge\nolimits_{d\in\{1,3,\ldots,k\}} \bigcirc\neg c_{\pi_d} \wedge \bigcirc c_{\pi'}\Bigg) \Rightarrow \\ \Bigg(\bigwedge\nolimits_{d\in\{2,4,\ldots,k+1\}} \bigcirc\neg c_{\pi_d} \wedge \\ \diamondsuit\Bigg[\bigvee\nolimits_{d\in[1,k+1]}\Big(\big(q^d_{\pi_d} \Leftrightarrow q^d_{\pi'}\big) \wedge \\ \big((p_{\pi'} \wedge p_{\pi_d}) \vee (\bar{p}_{\pi'} \wedge \bar{p}_{\pi_d})\big)\Big)\Bigg]\Bigg)\end{aligned}$$

Intuitively, φ_{map} expresses the following: for all the clause traces π' and all traces that valuate universally quantified variables ($\pi_1, \pi_3, \ldots$), there exist traces evaluating the existentially quantified variables ($\pi_2, \pi_4, \ldots$), where either p or $\bar{p}$ eventually matches its counterpart position in the clause trace π'. The dependencies between the trace quantifiers for the valuation of the variables match the dependencies in the quantified Boolean formula. A special case are the outermost existential variables in the QBF. The corresponding trace quantifier (for π_{k+1}) is universal, rather than existential. As a result, the formula has only $k-1$ alternations. We allow synthesis to reduce the valuations for the outermost existential variables to a single valuation. Hence, universal and existential quantification is the same. ■

Finally, Theorem 8 implies that the synthesis problem for acyclic plants and HyperLTL formulas with an arbitrary number of quantifiers is in PSPACE.

Corollary 3: CS[HyperLTL, acyclic] is in PSPACE in the size of the plant.

VII. Complexity of Controller Synthesis for General Graphs

In this section, we investigate the complexity of the controller synthesis problem for general graphs. We again begin with the alternation-free fragment and then continue with formulas with quantifier alternation.

A. The Alternation-free Fragment

We start with the existential fragment. As for acyclic graphs, the controller synthesis problem is already solved by model checking.

Theorem 9: CS[E*-HyperLTL, general] is NL-complete in the size of the plant.

Proof: Analogously to the proof of Theorem 5, we note that, for existential formulas, the synthesis problem is equivalent to the model checking problem. A given plant satisfies the formula if and only if it has a solution to the synthesis problem. If the formula is satisfied, then the solution is simply the original plant. Since the model checking problem for existential formulas for general graphs is NL-complete [21], the same holds for the synthesis problem. ■

Similar to tree-shaped and acyclic graphs, the synthesis problem for the universal fragment is also NP-complete.

Theorem 10: CS[A^+-HyperLTL, general] is NP-complete in the size of the plant.

Proof: For membership in NP, we nondeterministically guess a solution to the synthesis problem, and verify the correctness of the universally quantified HyperLTL formula against the solution in polynomial time in the size of the plant. NP-hardness follows from the NP-hardness of the synthesis problem for LTL [22]. ■

B. Formulas with Quantifier Alternation

Next, we consider formulas where the number of quantifier alternations is bounded by a constant k. We show that changing the frame structure from acyclic to general graphs results in a significant increase in complexity (see Table I).

Theorem 11: CS[E^*A^*-HyperLTL, general] is in PSPACE in the size of the plant. CS[A^*E^*-HyperLTL, general] is PSPACE-complete in the size of the plant. For $k \geq 2$, CS[$(EA)^k$-HyperLTL, general] and CS[$(AE)^k$-HyperLTL, general] are $(k-1)$-EXPSPACE-complete in the size of the plant.

Proof: The claimed complexities are those of the model checking problem [23]. For the upper bound, we guess, in PSPACE, a solution to the control problem and then verify, using the model checking algorithm for the considered fragment of HyperLTL, that the solution satisfies the HyperLTL formula.

For the lower bound, we reduce the model checking problem to the controller synthesis problem by identifying each transition of the given plant as an uncontrollable transition of the plant. In this way, the controller synthesis cannot modify the plant, and the synthesis succeeds iff the given plant in the model checking problem already satisfies the HyperLTL formula. ■

Finally, Theorem 11 implies that the repair problem for general plants and HyperLTL formulas with an arbitrary number of quantifiers is in NONELEMENTARY.

Corollary 4: CS[HyperLTL, general] is NONELEMENTARY in the size of the plant.

VIII. Related Work

There has been a lot of recent progress in automatically verifying [13], [21], [24], [25] and monitoring [16], [26]–[31] HyperLTL specifications. HyperLTL is also supported by a growing set of tools, including the model checker MCHyper [13], [21], the satisfiability checkers EAHyper [32] and MGHyper [33], and the runtime monitoring tool RVHyper [30].

The closest work to the study in this paper is the analysis of the *program repair* problem for HyperLTL [15]. The repair problem is to find a subset of traces of a Kripke structure that satisfies a given HyperLTL formula. Thus, the repair problem is similar to the controller synthesis problem studied in this paper. In both problems, the goal is to prune the set of transitions of the given plant or model. However, in program repair, all transitions are controllable, whereas in controller synthesis the pruning cannot be applied to uncontrollable transitions. We draw the following comparison and contrast between the results in [15] and this paper:

- The general positive result of this paper is that although controller synthesis is typically perceived as a more difficult problem than program repair (due to the existence of uncontrollable transitions), our study, summarized in Table I, shows that the complexity of controller synthesis and program repair for HyperLTL remain pretty close. This result may appear counterintuitive. For some cases, there is a simple explanation why the complexities are similar. For example, for general graphs, the complexity is dominated by the model checking complexity. Both problems can be solved by first guessing a solution and then verifying it. The complexity of the model checking problem (which is the dominating factor) is the same for both problems and as a result, the complexity of program repair and controller synthesis is the same (NONELEMENTARY). However, this is not always the case. For example, for acyclic graphs, the fact that guessing the controller is more difficult than guessing the repair leads to a higher complexity (within the polynomial hierarchy). Another interesting observation is the effect of the universal quantifiers in the $\forall\forall$ fragment. While the repair problem for the $\forall\forall$ fragment is L-complete for tree-shaped graphs, the problem becomes NP-complete for the controller synthesis problem. Also, while the repair problem for the $\forall\forall$ fragment is NL-complete for acyclic graphs, it becomes NP-complete for the controller synthesis problem. This is significant, because many important security policies such as certain types of noninterference [10] and observational determinism [34] fit in this fragment.
- Although the proof techniques in this paper are similar to those in [15], leveraging the existence of uncontrollable transitions has made our lower bound proofs more elegant. In particular, the proofs in [15] need to incorporate complex constraints in the HyperLTL formulas to make sure that during reductions, parts of the state space related to clauses of the input (e.g., SAT or QBF) formulas are not removed. Here, we mimic this in a more elegant way by using uncontrollable transitions, which cannot be removed during synthesis.

The controller synthesis problem studied in this paper is also related to classic *supervisory control*, where, for a given plant, a supervisor is constructed that selects an appropriate subset of the plant's controllable actions to ensure that the resulting behavior is safe [35]–[37].

Directly related to the controller synthesis problem studied in this paper is the *satisfiability*. The satisfiability problem for HyperLTL was shown to be decidable for the $\exists^*\forall^*$ fragment and for any fragment that includes a $\forall\exists$ quantifier alternation [38]. The hierarchy of hyperlogics beyond HyperLTL has been studied in [39].

The general *synthesis* problem differs from controller synthesis in that the solutions are not limited to the state graph of the plant. For HyperLTL, synthesis was shown to be undecidable in general, and decidable for the $\exists^*$ and

$\exists^*\forall$ fragments [12]. While the synthesis problem becomes, in general, undecidable as soon as there are two universal quantifiers, there is a special class of universal specifications, called the linear $\forall^*$-fragment, which is still decidable. The linear $\forall^*$-fragment corresponds to the decidable *distributed synthesis* problems [40]. The *bounded synthesis* problem [12], [13] considers only systems up to a given bound on the number of states. Bounded synthesis has been successfully applied to various benchmarks including the dining cryptographers [14].

The problem of *model checking* hyperproperties for tree-shaped and acyclic graphs was studied in [18]. Earlier, a similar study of the impact of structural restrictions on the complexity of the model checking problem has also been carried out for LTL [41].

Our motivating example draws from the substantial literature on specifying and verifying *non-repudiation protocols* [42]–[45]. In particular, [45] discusses the need for the consideration of incomplete information. We are not aware, however, of previous work on the automatic synthesis of trusted third parties for such protocols.

IX. Conclusion and Future Work

We have presented a rigorous classification of the complexity of the *controller synthesis* problem for *hyperproperties* expressed in HyperLTL. We considered general, acyclic, and tree-shaped plants. We showed that for trees, the complexity of the synthesis problem in the size of the plant does not go beyond NP. While the problem is complete for L for the alternation-free existential fragment, it is complete for NP for the alternation-free universal fragment. The problem is complete for P for the fragment with only one quantifier alternation, where the leading quantifier is universal. For acyclic plants, the complexity is in PSPACE (in the level of the polynomial hierarchy that corresponds to the number of quantifier alternations). The problem is NL-complete for the alternation-free existential fragment. Similar to trees, the problem is NP-complete for the alternation-free universal fragment. For general graphs, the problem is NONELEMENTARY for an arbitrary number of quantifier alternations. For a bounded number k of alternations, the problem is $(k-1)$-EXPSPACE-complete.

It is interesting to compare controller synthesis to program repair [18]. With the notable exception the universal fragment of HyperLTL for trees and acyclic graphs, the complexities of controller synthesis and program repair are largely aligned. This is mainly due to the fact that synthesizing a controller involves computing a subset of the controllable transitions such that the specification is satisfied. This is also the case for program repair, except that all transitions of the system are controllable.

As for future work, we plan to develop efficient controller synthesis algorithms for different fragments of HyperLTL along the lines of QBF-based synthesis methods for hyperproperties [12], [13]. It would furthermore be interesting to see if the differences we observed for HyperLTL carry over to other hyperlogics beyond HyperLTL (cf. [9], [39], [46], [47]).

Acknowledgments

This work is sponsored in part by the United States NSF SaTC Award 1813388. It was also supported by the German Research Foundation (DFG) as part of the Collaborative Research Center "Methods and Tools for Understanding and Controlling Privacy" (CRC 1223) and the Collaborative Research Center "Foundations of Perspicuous Software Systems" (TRR 248, 389792660), and by the European Research Council (ERC) Grant OSARES (No. 683300).

References

[1] E. A. Emerson and E. M. Clarke, "Using branching time temporal logic to synthesize synchronization skeletons," *Science of Computer Programming*, vol. 2(3), pp. 241–266, 1982.

[2] Z. Manna and P. Wolper, "Synthesis of communicating processes from temporal logic specifications," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 6(1), pp. 68–93, 1984.

[3] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[4] https://www.nytimes.com/2017/04/08/us/dallas-emergency-sirens-hacking.html.

[5] https://en.wikipedia.org/wiki/Heartbleed.

[6] https://en.wikipedia.org/wiki/Yahoo!_data_breaches.

[7] https://en.wikipedia.org/wiki/Unreachable_code.

[8] M. R. Clarkson and F. B. Schneider, "Hyperproperties," *Journal of Computer Security*, vol. 18, no. 6, pp. 1157–1210, 2010.

[9] M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez, "Temporal logics for hyperproperties," in *Proceedings of the 3rd Conference on Principles of Security and Trust POST*, 2014, pp. 265–284.

[10] J. A. Goguen and J. Meseguer, "Security policies and security models," in *IEEE Symp. on Security and Privacy*, 1982, pp. 11–20.

[11] D. McCullough, "Noninterference and the composability of security properties," in *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, 1988, pp. 177–186.

[12] B. Finkbeiner, C. Hahn, P. Lukert, M. Stenger, and L. Tentrup, "Synthesis from hyperproperties," *Acta Inf.*, vol. 57, no. 1, pp. 137–163, 2020.

[13] N. Coenen, B. Finkbeiner, C. Sánchez, and L. Tentrup, "Verifying hyperliveness," in *Proceedings of the 31st International Conference on Computer Aided Verification (CAV)*, 2019, pp. 121–139.

[14] D. Chaum, "Security without identification: Transaction systems to make big brother obsolete," *Communications of the ACM*, vol. 28, no. 10, pp. 1030–1044, 1985.

[15] B. Bonakdarpour and B. Finkbeiner, "Program repair for hyperproperties," in *Proceedings of the 17th Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2019, pp. 423–441.

[16] S. Stucki, C. Sánchez, G. Schneider, and B. Bonakdarpour, "Graybox monitoring of hyperproperties," in *Proceedings of the 23rd International Symposium on Formal Methods (FM)*, 2019, to appear.

[17] R. Alur and S. Tripakis, "Automatic synthesis of distributed protocols," *SIGACT News*, vol. 48, no. 1, pp. 55–90, 2017.

[18] B. Bonakdarpour and B. Finkbeiner, "The complexity of monitoring hyperproperties," in *Proceedings of the 31st IEEE Computer Security Foundations Symposium CSF*, 2018, pp. 162–174.

[19] K. Etessami, "Counting quantifiers, successor relations, and logarithmic space," *Journal of Compuer and System Sciences*, vol. 54, no. 3, pp. 400–411, 1997.

[20] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman, 1979.

[21] B. Finkbeiner, M. N. Rabe, and C. Sánchez, "Algorithms for model checking HyperLTL and HyperCTL*," in *Proceedings of the 27th International Conference on Computer Aided Verification (CAV)*, 2015, pp. 30–48.

[22] B. Bonakdarpour, A. Ebnenasir, and S. S. Kulkarni, "Complexity results in revising UNITY programs," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 1, pp. 1–28, January 2009.

[23] M. N. Rabe, "A temporal logic approach to information-flow control," Ph.D. dissertation, Saarland University, 2016.

[24] B. Finkbeiner, C. Müller, H. Seidl, and E. Zalinescu, "Verifying Security Policies in Multi-agent Workflows with Loops," in *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*, 2017.

[25] B. Finkbeiner, C. Hahn, and H. Torfah, "Model checking quantitative hyperproperties," in *Proceedings of the 30th International Conference on Computer Aided Verification*, 2018, pp. 144–163.

[26] S. Agrawal and B. Bonakdarpour, "Runtime verification of k-safety hyperproperties in HyperLTL," in *Proceedings of the IEEE 29th Computer Security Foundations (CSF)*, 2016, pp. 239–252.

[27] B. Finkbeiner, C. Hahn, M. Stenger, and L. Tentrup, "Monitoring hyperproperties," *Formal Methods in System Design (FMSD)*, vol. 54, no. 3, pp. 336–363, 2019.

[28] N. Brett, U. Siddique, and B. Bonakdarpour, "Rewriting-based runtime verification for alternation-free HyperLTL," in *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2017, pp. 77–93.

[29] B. Bonakdarpour, C. Sánchez, and G. Schneider, "Monitoring hyperproperties by combining static analysis and runtime verification," in *Proceedings of the 8th Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, 2018, pp. 8–27.

[30] B. Finkbeiner, C. Hahn, M. Stenger, and L. Tentrup, "RVHyper: A runtime verification tool for temporal hyperproperties," in *Proceedings of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2018, pp. 194–200.

[31] C. Hahn, M. Stenger, and L. Tentrup, "Constraint-based monitoring of hyperproperties," in *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2019, pp. 115–131.

[32] B. Finkbeiner, C. Hahn, and M. Stenger, "EAHyper: Satisfiability, implication, and equivalence checking of hyperproperties," in *Proceedings of the 29th International Conference on Computer Aided Verification (CAV)*, 2017, pp. 564–570.

[33] B. Finkbeiner, C. Hahn, and T. Hans, "MGHyper: Checking satisfiability of HyperLTL formulas beyond the $\exists^*\forall^*$ fragment," in *Proceedings of the 16th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2018, pp. 521–527.

[34] S. Zdancewic and A. C. Myers, "Observational determinism for concurrent program security," in *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW)*, 2003, p. 29.

[35] J. G. Thistle and W. M. Wonham, "Control problems in a temporal logic framework," *International Journal of Control*, vol. 44, no. 4, pp. 943–976, 1986.

[36] F. Lin, "Analysis and synthesis of discrete event systems using temporal logic," in *Proceedings of the 1991 IEEE International Symposium on Intelligent Control*, Aug 1991, pp. 140–145.

[37] S. Jiang and R. Kumar, "Supervisory control of discrete event systems with CTL* temporal logic specifications," *SIAM Journal on Control and Optimization*, vol. 44, no. 6, pp. 2079–2103, 2006.

[38] B. Finkbeiner and C. Hahn, "Deciding hyperproperties," in *Proceedings of the 27th International Conference on Concurrency Theory (CONCUR)*, 2016, pp. 13:1–13:14.

[39] N. Coenen, B. Finkbeiner, C. Hahn, and J. Hofmann, "The hierarchy of hyperlogics," in *Proceedings 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2019, pp. 1–13.

[40] B. Finkbeiner and S. Schewe, "Uniform distributed synthesis," in *Proceedings of the 20th ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2005, pp. 321–330.

[41] L. Kuhtz and B. Finkbeiner, "Weak Kripke structures and LTL," in *Proceedings of the 22nd International Conference on Concurrency Theory (CONCUR)*, 2011, pp. 419–433.

[42] P. D. Ezhilchelvan and S. K. Shrivastava, "Systematic development of a family of fair exchange protocols," in *Data and Applications Security XVII: Status and Prospects, IFIP TC-11 WG 11.3 Seventeenth Annual Working Conference on Data and Application Security, August4-6, 2003, Estes Park, Colorado, USA*, 2003, pp. 243–258.

[43] P. Liu, P. Ning, and S. Jajodia, "Avoiding loss of fairness owing to process crashes in fair data exchange protocols," in *2000 International Conference on Dependable Systems and Networks (DSN 2000) (formerly FTCS-30 and DCCA-8), 25-28 June 2000, New York, NY, USA*, 2000, pp. 631–640.

[44] S. Kremer and J.-F. Raskin, "A game-based verification of non-repudiation and fair exchange protocols," in *CONCUR 2001 — Concurrency Theory*, K. G. Larsen and M. Nielsen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 551–565.

[45] W. Jamroga, S. Mauw, and M. Melissen, "Fairness in non-repudiation protocols," in *Security and Trust Management*, C. Meadows and C. Fernandez-Gago, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 122–139.

[46] B. Finkbeiner, C. Müller, H. Seidl, and E. Zalinescu, "Verifying Security Policies in Multi-agent Workflows with Loops," in *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*, 2017, pp. 633–645.

[47] E. Ábrahám and B. Bonakdarpour, "HyperPCTL: A temporal logic for probabilistic hyperproperties," in *Proceedings of the 15th International Conference on Quantitative Evaluation of Systems (QEST)*, 2018, pp. 20–35.