Synthesizing Reactive Systems from Hyperproperties

Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup(✉)

Reactive Systems Group, Saarland University,
Saarbrücken, Germany
{finkbeiner,hahn,lukert,
stenger,tentrup}@react.uni-saarland.de

Abstract. We study the reactive synthesis problem for hyperproperties given as formulas of the temporal logic HyperLTL. Hyperproperties generalize trace properties, i.e., sets of traces, to *sets of sets* of traces. Typical examples are information-flow policies like noninterference, which stipulate that no sensitive data must leak into the public domain. Such properties cannot be expressed in standard linear or branching-time temporal logics like LTL, CTL, or CTL*. We show that, while the synthesis problem is undecidable for full HyperLTL, it remains decidable for the $\exists^*$, $\exists^*\forall^1$, and the *linear* $\forall^*$ fragments. Beyond these fragments, the synthesis problem immediately becomes undecidable. For universal HyperLTL, we present a semi-decision procedure that constructs implementations and counterexamples up to a given bound. We report encouraging experimental results obtained with a prototype implementation on example specifications with hyperproperties like symmetric responses, secrecy, and information-flow.

1 Introduction

Hyperproperties [5] generalize trace properties in that they not only check the correctness of *individual* computation traces in isolation, but relate *multiple* computation traces to each other. HyperLTL [4] is a logic for expressing temporal hyperproperties, by extending linear-time temporal logic (LTL) with *explicit* quantification over traces. HyperLTL has been used to specify a variety of information-flow and security properties. Examples include classical properties like non-interference and observational determinism, as well as quantitative information-flow properties, symmetries in hardware designs, and formally verified error correcting codes [12]. For example, observational determinism can be expressed as the HyperLTL formula $\forall\pi\forall\pi'.\square(I_\pi = I_{\pi'}) \rightarrow \square(O_\pi = O_{\pi'})$, stating that, for every pair of traces, if the observable inputs are the same, then the observable outputs must be same as well. While the satisfiability [9], model checking [4,12], and runtime verification [1,10] problem for HyperLTL has been studied, the *reactive synthesis* problem of HyperLTL is, so far, still open.

Supported by the European Research Council (ERC) Grant OSARES (No. 683300).

H. Chockler and G. Weissenbacher (Eds.): CAV 2018, LNCS 10981, pp. 289–306, 2018.
https://doi.org/10.1007/978-3-319-96145-3_16

In reactive synthesis, we automatically construct an implementation that is guaranteed to satisfy a given specification. A fundamental difference to verification is that there is no human programmer involved: in verification, the programmer would first produce an implementation, which is then verified against the specification. In synthesis, the implementation is directly constructed from the specification. Because there is no programmer, it is crucial that the specification contains *all* desired properties of the implementation: the synthesized implementation is guaranteed to satisfy the given specification, but nothing is guaranteed beyond that. The added expressive power of HyperLTL over LTL is very attractive for synthesis: with synthesis from hyperproperties, we can guarantee that the implementation does not only accomplish the desired functionality, but is also free of information leaks, is symmetric, is fault-tolerant with respect to transmission errors, etc.

More formally, the reactive synthesis problem asks for a *strategy*, that is a tree branching on environment inputs whose nodes are labeled by the system output. Collecting the inputs and outputs along a branch of the tree, we obtain a trace. If the set of traces collected from the branches of the strategy tree satisfies the specification, we say that the strategy *realizes* the specification. The specification is *realizable* iff there exists a strategy tree that realizes the specification. With LTL specifications, we get trees where the trace on each individual branch satisfies the LTL formula. With HyperLTL, we additionally get trees where the traces between different branches are in a specified relationship. This is dramatically more powerful.

Consider, for example, the well-studied *distributed* version of the reactive synthesis problem, where the system is split into a set of processes, that each only see a subset of the inputs. The distributed synthesis problem for LTL can be expressed as the standard (non-distributed) synthesis problem for HyperLTL, by adding for each process the requirement that the process output is *observationally deterministic* in the process input. HyperLTL synthesis thus subsumes distributed synthesis. The information-flow requirements realized by HyperLTL synthesis can, however, be much more sophisticated than the observational determinism needed for distributed synthesis. Consider, for example, the *dining cryptographers* problem [3]: three cryptographers C_a, C_b, and C_c sit at a table in a restaurant having dinner and either one of cryptographers or, alternatively, the NSA must pay for their meal. Is there a protocol where each cryptographer can find out whether it was a cryptographer who paid or the NSA, but cannot find out which cryptographer paid the bill?

Synthesis from LTL formulas is known to be decidable in doubly exponential time. The fact that the distributed synthesis problem is undecidable [21] immediately eliminates the hope for a similar general result for HyperLTL. However, since LTL is obviously a fragment of HyperLTL, this immediately leads to the question whether the synthesis problem is still decidable for fragments of HyperLTL that are close to LTL but go beyond LTL: when exactly does the synthesis problem become undecidable? From a more practical point of view, the interesting question is whether semi-algorithms for distributed synthesis [7,14], which have been successful in constructing distributed systems from LTL specifications despite the undecidability of the general problem, can be extended to HyperLTL?

In this paper, we answer the first question by studying the $\exists^*$, $\exists^*\forall^1$, and the *linear* $\forall^*$ fragment. We show that the synthesis problem for all three fragments is decidable, and the problem becomes undecidable as soon as we go beyond these fragments. In particular, the synthesis problem for the full $\forall^*$ fragment, which includes observational determinism, is undecidable.

We answer the second question by studying the *bounded* version of the synthesis problem for the $\forall^*$ fragment. In order to detect realizability, we ask whether, for a universal HyperLTL formula φ and a given bound n on the number of states, there exists a representation of the strategy tree as a finite-state machine with no more than n states that satisfies φ. To detect unrealizability, we check whether there exists a counterexample to realizability of bounded size. We show that both checks can be effectively reduced to SMT solving.

Related Work. HyperLTL [4] is a successor of the temporal logic SecLTL [6] used to characterize temporal information-flow. The model-checking [4,12], satisfiability [9], monitoring problem [1,10], and the first-order extension [17] of HyperLTL has been studied before. To the best of the authors knowledge, this is the first work that considers the synthesis problem for temporal hyperproperties. We base our algorithms on well-known synthesis algorithms such as bounded synthesis [14] that itself is an instance of Safraless synthesis [18] for ω-regular languages. Further techniques that we adapt for hyperproperties are lazy synthesis [11] and the bounded unrealizability method [15,16].

Hyperproperties [5] can be seen as a unifying framework for many different properties of interest in multiple distinct areas of research. Information-flow properties in security and privacy research are hyperproperties [4]. HyperLTL subsumes logics that reason over knowledge [4]. Information-flow in distributed systems is another example of hyperproperties, and the HyperLTL realizability problem subsumes both the distributed synthesis problem [13,21] as well as synthesis of fault-tolerant systems [16]. In circuit verification, the semantic independence of circuit output signals on a certain set of inputs, enabling a range of potential optimizations, is a hyperproperty.

2 Preliminaries

HyperLTL. HyperLTL [4] is a temporal logic for specifying hyperproperties. It extends LTL by quantification over trace variables π and a method to link atomic propositions to specific traces. The set of trace variables is $\mathcal{V}$. Formulas in HyperLTL are given by the grammar

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

where $a \in \text{AP}$ and $\pi \in \mathcal{V}$. The alphabet of a HyperLTL formula is 2^{AP}. We allow the standard boolean connectives $\wedge$, $\rightarrow$, $\leftrightarrow$ as well as the derived LTL operators release $\varphi \mathcal{R} \psi \equiv \neg(\neg\varphi \mathcal{U} \neg\psi)$, eventually $\diamond \varphi \equiv \mathit{true} \mathcal{U} \varphi$, globally $\square \varphi \equiv \neg \diamond \neg\varphi$, and weak until $\varphi \mathcal{W} \psi \equiv \square\varphi \vee (\varphi \mathcal{U} \psi)$.

The semantics is given by the satisfaction relation $\vDash_T$ over a set of traces $T \subseteq (2^{\mathrm{AP}})^\omega$. We define an assignment $\Pi : \mathcal{V} \to (2^{\mathrm{AP}})^\omega$ that maps trace variables to traces. $\Pi[i,\infty]$ is the trace assignment that is equal to $\Pi(\pi)[i,\infty]$ for all π and denotes the assignment where the first i items are removed from each trace.

$$\begin{array}{ll}
\Pi \vDash_T a_\pi & \text{if } a \in \Pi(\pi)[0] \\
\Pi \vDash_T \neg\varphi & \text{if } \Pi \nvDash_T \varphi \\
\Pi \vDash_T \varphi \vee \psi & \text{if } \Pi \vDash_T \varphi \text{ or } \Pi \vDash_T \psi \\
\Pi \vDash_T \bigcirc \varphi & \text{if } \Pi[1,\infty] \vDash_T \varphi \\
\Pi \vDash_T \varphi \mathcal{U} \psi & \text{if } \exists i \geq 0.\, \Pi[i,\infty] \vDash_T \psi \wedge \forall 0 \leq j < i.\, \Pi[j,\infty] \vDash_T \varphi \\
\Pi \vDash_T \exists \pi.\, \varphi & \text{if there is some } t \in T \text{ such that } \Pi[\pi \mapsto t] \vDash_T \varphi \\
\Pi \vDash_T \forall \pi.\, \varphi & \text{if for all } t \in T \text{ holds that } \Pi[\pi \mapsto t] \vDash_T \varphi
\end{array}$$

We write $T \vDash \varphi$ for $\{\} \vDash_T \varphi$ where $\{\}$ denotes the empty assignment. Two HyperLTL formulas φ and ψ are equivalent, written $\varphi \equiv \psi$ if they have the same models.

(In)dependence is a common hyperproperty for which we define the following syntactic sugar. Given two disjoint subsets of atomic propositions $C \subseteq \mathrm{AP}$ and $A \subseteq \mathrm{AP}$, we define independence as the following HyperLTL formula

$$D_{A \mapsto C} := \forall\pi\forall\pi'.\, \left(\bigvee_{a \in A} (a_\pi \nleftrightarrow a_{\pi'})\right) \mathcal{R} \left(\bigwedge_{c \in C} (c_\pi \leftrightarrow c_{\pi'})\right) \quad . \tag{1}$$

This guarantees that every proposition $c \in C$ solely depends on propositions A.

Strategies. A *strategy* $f \colon (2^I)^* \to 2^O$ maps sequences of input valuations 2^I to an output valuation 2^O. The behavior of a strategy $f \colon (2^I)^* \to 2^O$ is characterized by an infinite tree that branches by the valuations of I and whose nodes $w \in (2^I)^*$ are labeled with the strategic choice $f(w)$. For an infinite word $w = w_0 w_1 w_2 \cdots \in (2^I)^\omega$, the corresponding labeled path is defined as $(f(\epsilon) \cup w_0)(f(w_0) \cup w_1)(f(w_0 w_1) \cup w_2) \cdots \in (2^{I \cup O})^\omega$. We lift the set containment operator $\in$ to the containment of a labeled path $w = w_0 w_1 w_2 \cdots \in (2^{I \cup O})^\omega$ in a strategy tree induced by $f \colon (2^I)^* \to 2^O$, i.e., $w \in f$ if, and only if, $f(\epsilon) = w_0 \cap O$ and $f((w_0 \cap I) \cdots (w_i \cap I)) = w_{i+1} \cap O$ for all $i \geq 0$. We define the satisfaction of a HyperLTL formula φ (over propositions $I \cup O$) on strategy f, written $f \vDash \varphi$, as $\{w \mid w \in f\} \vDash \varphi$. Thus, a strategy f is a model of φ if the set of labeled paths of f is a model of φ.

3 HyperLTL Synthesis

In this section, we identify fragments of HyperLTL for which the realizability problem is decidable. Our findings are summarized in Table 1.

Definition 1 (HyperLTL Realizability). *A HyperLTL formula φ over atomic propositions $AP = I \dot\cup O$ is realizable if there is a strategy $f \colon (2^I)^* \to 2^O$ that satisfies φ.*

Table 1. Summary of decidability results.

$\exists^*$	$\exists^*\forall^1$	$\exists^*\forall^{>1}$	$\forall^*$	$\forall^*\exists^*$	*linear* $\forall^*$
PSpace-complete	3ExpTime	undecidable			decidable

We base our investigation on the structure of the quantifier prefix of the HyperLTL formulas. We call a HyperLTL formula φ (quantifier) *alternation-free* if the quantifier prefix consists solely of either universal or existential quantifiers. We denote the corresponding fragments as the (universal) $\forall^*$ and the (existential) $\exists^*$ fragment, respectively. A HyperLTL formula is in the $\exists^*\forall^*$ fragment, if it starts with arbitrarily many existential quantifiers, followed by arbitrarily many universal quantifiers. Respectively for the $\forall^*\exists^*$ fragment. For a given natural number n, we refer to a bounded number of quantifiers with $\forall^n$, respectively $\exists^n$. The $\forall^1$ realizability problem is equivalent to the LTL realizability problem.

$\exists^*$ Fragment. We show that the realizability problem for existential HyperLTL is PSpace-complete. We reduce the realizability problem to the satisfiability problem for bounded one-alternating $\exists^*\forall^2$HyperLTL [9], i.e., finding a trace set T such that $T \vDash \varphi$.

Lemma 1. *An existential HyperLTL formula φ is realizable if, and only if, $\psi :=$ $\varphi \wedge D_{I \mapsto O}$ is satisfiable.*

Proof. Assume $f\colon (2^I)^* \rightarrow 2^O$ realizes φ, that is $f \vDash \varphi$. Let $T = \{w \mid w \in f\}$ be the set of traces generated by f. It holds that $T \vDash \varphi$ and $T \vDash D_{I \mapsto O}$. Therefore, ψ is satisfiable. Assume ψ is satisfiable. Let S be a set of traces that satisfies ψ. We construct a strategy $f\colon (2^I)^* \rightarrow 2^O$ as

$$f(\sigma) = \begin{cases} w_{|\sigma|} \cap O & \text{if } \sigma \text{ is a prefix of some } w|_I \text{ with } w \in S \text{ , and} \\ \emptyset & \text{otherwise .} \end{cases}$$

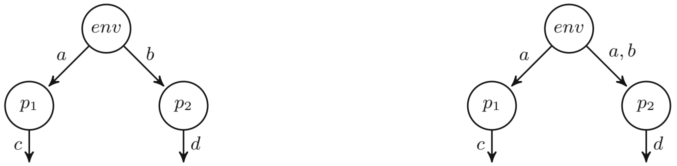
where $w|_I$ denotes the trace restricted to I, formally $w_i \cap I$ for all $i \geq 0$. Note that if there are multiple candidates $w \in S$, then $w_{|\sigma|} \cap O$ is the same for all of them because of the required non-determinism $D_{I \mapsto O}$. By construction, all traces in S are contained in f and with $S \vDash \varphi$ it holds that $f \vDash \varphi$ as φ is an existential formula.

Theorem 1. *Realizability of existential HyperLTL specifications is decidable.*

Proof. The formula ψ from Lemma 1 is in the $\exists^*\forall^2$ fragment, for which satisfiability is decidable [9].

Corollary 1. *Realizability of $\exists^*$ HyperLTL specifications is* PSpace*-complete.*

Proof. Given an existential HyperLTL formula, we gave a linear reduction to the satisfiability of the $\exists^*\forall^2$ fragment in Lemma 1. The satisfiability problem for a bounded number of universal quantifiers is in PSpace [9]. Hardness follows from LTL satisfiability, which is equivalent to the $\exists^1$ fragment.



(a) An architecture of two processes that specify process p_1 to produce c from a and p_2 to produce d from b.

(b) The same architecture as on the left, where only the inputs of process p_2 are changed to a and b.

Fig. 1. Distributed architectures

$\forall^*$ Fragment. In the following, we will use the *distributed synthesis* problem, i.e., the problem whether there is an implementation of processes in a distributed *architecture* that satisfies an LTL formula. Formally, a distributed architecture A is a tuple $\langle P, p_{env}, \mathcal{I}, \mathcal{O}\rangle$ where P is a finite set of processes with distinguished environment process $p_{env} \in P$. The functions $\mathcal{I}\colon P \to 2^{\mathrm{AP}}$ and $\mathcal{O}\colon P \to 2^{\mathrm{AP}}$ define the inputs and outputs of processes. While processes may share the same inputs (in case of broadcasting), the outputs of processes must be pairwise disjoint, i.e., for all $p \neq p' \in P$ it holds that $\mathcal{O}(p) \cap \mathcal{O}(p') = \emptyset$. W.l.o.g. we assume that $\mathcal{I}(p_{env}) = \emptyset$. The distributed synthesis problem for architectures without *information forks* [13] is decidable. Example architectures are depicted in Fig. 1. The architecture in Fig. 1a contains an information fork while the architecture in Fig. 1b does not. Furthermore, the processes in Fig. 1b can be ordered linearly according to the subset relation on the inputs.

Theorem 2. *The synthesis problem for universal HyperLTL is undecidable.*

Proof. In the $\forall^*$ fragment (and thus in the $\exists^*\forall^*$ fragment), we can encode a distributed architecture [13], for which LTL synthesis is undecidable. In particular, we can encode the architecture shown in Fig. 1a. This architecture basically specifies c to depend only on a and analogously d on b. That can be encoded by $D_{\{a\}\mapsto\{c\}}$ and $D_{\{b\}\mapsto\{d\}}$. The LTL synthesis problem for this architecture is already shown to be undecidable [13], i.e., given an LTL formula over $I = \{a, b\}$ and $O = \{c, d\}$, we cannot automatically construct processes p_1 and p_2 that realize the formula.

Linear $\forall^*$ Fragment. For characterizing the linear fragment of HyperLTL, we will present a transformation from a formula with arbitrarily many universal quantifiers to a formula with only one quantifier. This transformation collapses the universal quantifier into a single one and renames the path variables accordingly. For example, $\forall\pi_1\forall\pi_2.\,\square\, a_{\pi_1} \vee \square\, a_{\pi_2}$ is transformed into an equivalent $\forall^1$ formula $\forall\pi.\,\square\, a_\pi \vee \square\, a_\pi$. However, this transformation does not always produce equivalent formulas as $\forall\pi_1\forall\pi_2.\,\square(a_{\pi_1} \leftrightarrow a_{\pi_2})$ is not equivalent to its collapsed form $\forall\pi.\,\square(a_\pi \leftrightarrow a_\pi)$. Let φ be $\forall\pi_1 \cdots \forall\pi_n.\,\psi$. We define the collapsed formula of φ as $\mathit{collapse}(\varphi) \coloneqq \forall\pi.\,\psi[\pi_1 \mapsto \pi][\pi_2 \mapsto \pi]\ldots[\pi_n \mapsto \pi]$ where $\psi[\pi_i \mapsto \pi]$ replaces all occurrences of π_i in ψ with π. Although the collapsed term is not always equivalent to the original formula, we can use it as an indicator whether it is possible at all to express a universal formula with only one quantifier as stated in the following lemma.

Lemma 2. *Either $\varphi \equiv \mathit{collapse}(\varphi)$ or φ has no equivalent $\forall^1$ formula.*

Proof. Suppose there is some $\psi \in \forall^1$ with $\psi \equiv \varphi$. We show that $\psi \equiv \mathit{collapse}(\varphi)$. Let T be an arbitrary set of traces. Let $\mathcal{T} = \{\{w\} \mid w \in T\}$. Because $\psi \in \forall^1$, $T \vDash \psi$ is equivalent to $\forall T' \in \mathcal{T}.T' \vDash \psi$, which is by assumption equivalent to $\forall T' \in \mathcal{T}.T' \vDash \varphi$. Now, φ operates on singleton trace sets only. This means that all quantified paths have to be the same, which yields that we can use the same path variable for all of them. So $\forall T' \in \mathcal{T}.T' \vDash \varphi \leftrightarrow T' \vDash \mathit{collapse}(\varphi)$ that is again equivalent to $T \vDash \mathit{collapse}(\varphi)$. Because $\psi \equiv \mathit{collapse}(\varphi)$ and $\psi \equiv \varphi$ it holds that $\varphi \equiv \mathit{collapse}(\varphi)$.

The LTL realizability problem for distributed architectures without information forks [13] are decidable. These architectures are in some way *linear*, i.e., the processes can be ordered such that lower processes always have a subset of the information of upper processes. The linear fragment of universal HyperLTL addresses exactly these architectures.

In the following, we sketch the characterization of the linear fragment of HyperLTL. Given a formula φ, we seek for variable dependencies of the form $D_{J \mapsto \{o\}}$ with $J \subseteq I$ and $o \in O$ in the formula. If the part of the formula φ that relates multiple paths consists only of such constraints $D_{J \mapsto \{o\}}$ with the rest being an LTL property, we can interpret φ as a description of a distributed architecture. If furthermore, the $D_{J_i \mapsto \{o_i\}}$ constraints can be ordered such that $J_i \subseteq J_{i+1}$ for all i, the architecture is linear. There are three steps to check whether φ is in the linear fragment:

1. First, we have to add input-determinism to the formula $\varphi_{det} \coloneqq \varphi \wedge D_{I \mapsto O}$. This preserves realizability as strategies are input-deterministic.
2. Find for each output variable $o_i \in O$ possible sets of variables J_i, o_i depends on, such that $J_i \subseteq J_{i+1}$. To check whether the choice of J's is correct, we test if $\mathit{collapse}(\varphi) \wedge \bigwedge_{o_i \in O} D_{J_i \mapsto \{o_i\}}$ is equivalent to φ_{det}. This equivalence check is decidable as both formulas are in the universal fragment [9].
3. Finally, we construct the corresponding distributed realizability problem with linear architecture. Formally, we define the distributed architecture $A = \langle P, p_{env}, \mathcal{I}, \mathcal{O} \rangle$ with $P = \{p_i \mid o_i \in O\} \cup \{p_{env}\}$, $\mathcal{I}(p_i) = J_i$, $\mathcal{O}(p_i) = \{o_i\}$, and $\mathcal{O}(p_{env}) = I$. The LTL specification for the distributed synthesis problem is $\mathit{collapse}(\varphi)$.

Definition 2 (linear fragment of $\forall^*$). *A formula φ is in the linear fragment of $\forall^*$ iff for all $o_i \in O$ there is a $J_i \subseteq I$ such that $\varphi \wedge D_{I \mapsto O} \equiv \mathit{collapse}(\varphi) \wedge \bigwedge_{o_i \in O} D_{J_i \mapsto \{o_i\}}$ and $J_i \subseteq J_{i+1}$ for all i.*

Note, that each $\forall^1$ formula φ (or φ is collapsible to a $\forall^1$ formula) is in the linear fragment because we can set all $J_i = I$ and additionally $\mathit{collapse}(\varphi) = \varphi$ holds.

As an example of a formula in the linear fragment of $\forall^*$, consider $\varphi = \forall\pi, \pi'. D_{\{a\} \mapsto \{c\}} \wedge \square(c_\pi \leftrightarrow d_\pi) \wedge \square(b_\pi \leftrightarrow \bigcirc e_\pi)$ with $I = \{a, b\}$ and $O = \{c, d, e\}$. The corresponding formula asserting input-deterministism is $\varphi_{det} = \varphi \wedge D_{I \mapsto O}$. One possible choice of J's is $\{a, b\}$ for c, $\{a\}$ for d and $\{a, b\}$ for e. Note, that one can use either $\{a, b\}$ or $\{a\}$ for c as $D_{\{a\} \mapsto \{d\}} \wedge (c_\pi \leftrightarrow d_\pi)$ implies $D_{\{a\} \mapsto \{c\}}$. However, the apparent alternative $\{b\}$ for e would yield an undecidable architecture. It holds that φ_{det} and $collapse(\varphi) \wedge D_{\{a,b\} \mapsto \{c\}} \wedge D_{\{a\} \mapsto \{d\}} \wedge D_{\{a,b\} \mapsto \{e\}}$ are equivalent and, thus, that φ is in the linear fragment.

Theorem 3. *The linear fragment of universal HyperLTL is decidable.*

Proof. It holds that $\varphi \equiv collapse(\varphi) \wedge \bigwedge_{o_i \in O} D_{J_i \mapsto \{o_i\}}$ for some J_i's. The LTL distributed realizability problem for $collapse(\varphi)$ in the constructed architecture A is equivalent to the HyperLTL realizability of φ as the architecture A represents exactly the input-determinism represented by formula $\bigwedge_{o_i \in O} D_{J_i \mapsto \{o_i\}}$. The architecture is linear and, thus, the realizability problem is decidable.

$\exists^*\forall^1$ Fragment. In this fragment, we consider arbitrary many existential path quantifier followed by a single universal path quantifier. This fragment turns out to be still decidable. We solve the realizability problem for this fragment by reducing it to a decidable fragment of the distributed realizability problem.

Theorem 4. *Realizability of $\exists^*\forall^1$ HyperLTL specifications is decidable.*

Proof. Let φ be $\exists\pi_1 \ldots \exists\pi_n \forall\pi'. \psi$. We reduce the realizability problem of φ to the distributed realizability problem for LTL. For every existential path quantifier π_i, we introduce a copy of the atomic propositions, written a_{π_i} for $a \in$ AP. Intuitively, those select the paths in the strategy tree where the existential path quantifiers are evaluated. Thus, those propositions (1) have to encode an actual path in the strategy tree and (2) may not depend on the branching of the strategy tree. To ensure (1), we add the LTL constraint $\square(I_{\pi_i} = I_{\pi'}) \rightarrow \square(O_{\pi_i} = O_{\pi'})$ that asserts that if the inputs correspond to some path in the strategy tree, the outputs on those paths have to be the same. Property (2) is guaranteed by the distributed architecture, the processes generating the propositions a_{π_i} do not depend on the environment output. The resulting architecture A_φ is $\langle \{p_{env}, p, p'\}, p_{env}, \{p \mapsto \emptyset, p' \mapsto I_{\pi'}\}, \{p_{env} \mapsto I_{\pi'}, p \mapsto \bigcup_{1 \leq i \leq n} O_{\pi_i} \cup I_{\pi_i}, p' \mapsto O_{\pi'}\}\rangle$. It is easy to verify that A_φ does not contain an information fork, thus the realizability problem is decidable. The LTL specification θ is $\psi \wedge \bigwedge_{1 \leq i \leq n} \square(I_{\pi_i} = I_{\pi'}) \rightarrow \square(O_{\pi_i} = O_{\pi'})$. The implementation of process p' (if it exists) is a model for the HyperLTL formula (process p producing witness for the $\exists$ quantifier). Conversely, a model for φ can be used as an implementation of p'. Thus, the distributed synthesis problem $\langle A_\varphi, \theta\rangle$ has a solution if, and only if, φ is realizable.

$\forall^*\exists^*$ Fragment. The last fragment to consider are formulas in the $\forall^*\exists^*$ fragment. Whereas the $\exists^*\forall^1$ fragment remains decidable, the realizability problem of $\forall^*\exists^*$ turns out to be undecidable even when restricted to only one quantifier of both sorts ($\forall^1\exists^1$).

Theorem 5. *Realizability of* $\forall^*\exists^*$ *HyperLTL is undecidable.*

Proof. The proof is done via reduction from Post's Correspondence Problem (PCP) [22]. The basic idea follows the proof in [9].

4 Bounded Realizability

We propose an algorithm to synthesize strategies from specifications given in universal HyperLTL by searching for finite generators of realizing strategies. We encode this search as a satisfiability problem for a decidable constraint system.

Transition Systems. A *transition system* $\mathcal{S}$ is a tuple $\langle S, s_0, \tau, l\rangle$ where S is a finite set of states, $s_0 \in S$ is the designated initial state, $\tau\colon S \times 2^I \to S$ is the transition function, and $l\colon S \to 2^O$ is the state-labeling or output function. We generalize the transition function to sequences over 2^I by defining $\tau^*\colon (2^I)^* \to S$ recursively as $\tau^*(\epsilon) = s_0$ and $\tau^*(w_0 \cdots w_{n-1} w_n) = \tau(\tau^*(w_0 \cdots w_{n-1}), w_n)$ for $w_0 \cdots w_{n-1} w_n \in (2^I)^+$. A transition system $\mathcal{S}$ *generates* the strategy f if $f(w) = l(\tau^*(w))$ for every $w \in (2^I)^*$. A strategy f is called *finite-state* if there exists a transition system that generates f.

Overview. We first sketch the synthesis procedure and then proceed with a description of the intermediate steps. Let φ be a universal HyperLTL formula $\forall\pi_1 \cdots \forall\pi_n.\, \psi$. We build the automaton $\mathcal{A}_\psi$ whose language is the set of tuples of traces that satisfy ψ. We then define the acceptance of a transition system $\mathcal{S}$ on $\mathcal{A}_\psi$ by means of the self-composition of $\mathcal{S}$. Lastly, we encode the existence of a transition system accepted by $\mathcal{A}_\psi$ as an SMT constraint system.

Example 1. Throughout this section, we will use the following (simplified) running example. Assume we want to synthesize a system that keeps decisions secret until it is allowed to publish. Thus, our system has three input signals *decision*, indicating whether a decision was made, the secret *value*, and a signal to *publish* results. Furthermore, our system has two outputs, a *high* output *internal* that stores the value of the last decision, and a *low* output *result* that indicates the result. No information about decisions should be inferred until publication. To specify the functionality, we propose the LTL specification

$$\begin{aligned} &\square(\mathit{decision} \rightarrow (\mathit{value} \leftrightarrow \bigcirc\, \mathit{internal})) \\ \wedge\ &\square(\neg \mathit{decision} \rightarrow (\mathit{internal} \leftrightarrow \bigcirc\, \mathit{internal})) \\ \wedge\ &\square(\mathit{publish} \rightarrow \bigcirc(\mathit{internal} \leftrightarrow \mathit{result}))\ . \end{aligned} \tag{2}$$

The solution produced by the LTL synthesis tool BoSy [8], shown in Fig. 2, clearly violates our intention that results should be secret until publish: Whenever a decision is made, the output *result* changes as well.

We formalize the property that no information about the decision can be inferred from *result* until publication as the HyperLTL formula

$$\forall\pi\forall\pi'.\, (\mathit{publish}_\pi \vee \mathit{publish}_{\pi'})\ \mathcal{R}\ (\mathit{result}_\pi \leftrightarrow \mathit{result}_{\pi'})\ . \tag{3}$$

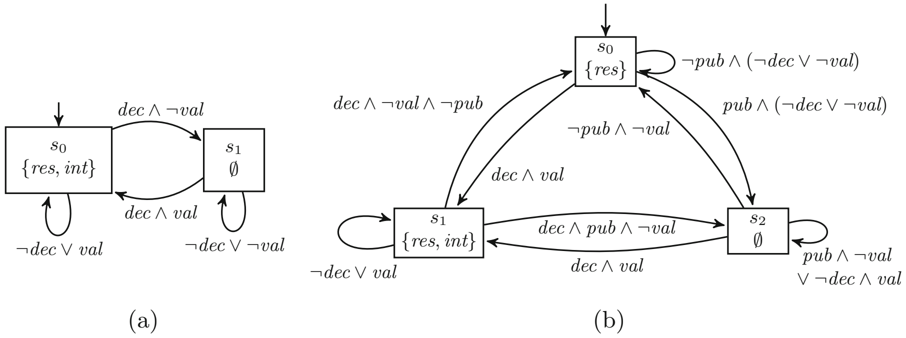


Fig. 2. Synthesized solutions for Example 1.

It asserts that for every pair of traces, the *result* signals have to be the same until (if ever) there is a *publish* signal on either trace. A solution satisfying both, the functional specification and the hyperproperty, is shown in Fig. 2. The system switches states whenever there is a decision with a different value than before and only exposes the decision in case there is a prior publish command.

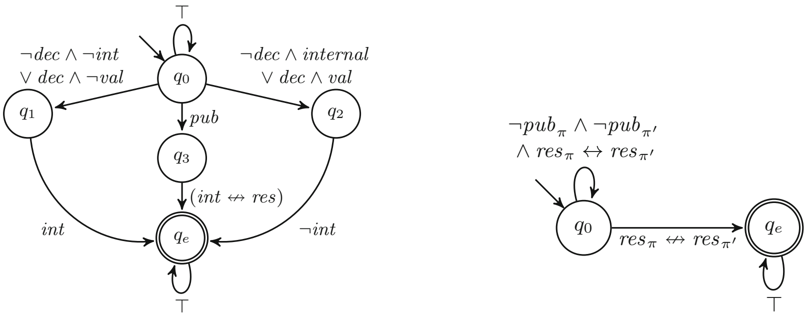
We proceed with introducing the necessary preliminaries for our algorithm.

Automata. A universal co-Büchi automaton $\mathcal{A}$ over a finite alphabet Σ is a tuple $\langle Q, q_0, \delta, F \rangle$, where Q is a finite set of states, $q_0 \in Q$ is the designated initial state, $\delta : Q \times 2^{\Sigma} \times Q$ is the transition relation, and $F \subseteq Q$ is the set of rejecting states. Given an infinite word $\sigma = \sigma_0 \sigma_1 \sigma_2 \cdots \in (2^{\Sigma})^{\omega}$, a run of σ on $\mathcal{A}$ is an infinite path $q_0 q_1 q_2 \cdots \in Q^{\omega}$ where for all $i \geq 0$ it holds that $(q_i, \sigma_i, q_{i+1}) \in \delta$. A run is accepting, if it contains only finitely many rejecting states. $\mathcal{A}$ accepts a word σ, if *all* runs of σ on $\mathcal{A}$ are accepting. The language of $\mathcal{A}$, written $\mathcal{L}(\mathcal{A})$, is the set $\{\sigma \in (2^{\Sigma})^{\omega} \mid \mathcal{A} \text{ accepts } \sigma\}$. We represent automata as directed graphs with vertex set Q and a symbolic representation of the transition relation δ as propositional boolean formulas $\mathbb{B}(\Sigma)$. The rejecting states in F are marked by double lines. The automata for the LTL and HyperLTL specifications from Example 1 are depicted in Fig. 3.

Run Graph. The run graph of a transition system $\mathcal{S} = \langle S, s_0, \tau, l \rangle$ on a universal co-Büchi automaton $\mathcal{A} = \langle Q, q_0, \delta, F \rangle$ is a directed graph $\langle V, E \rangle$ where $V = S \times Q$ is the set of vertices and $E \subseteq V \times V$ is the edge relation with

$$((s, q), (s', q')) \in E \text{ iff}$$
$$\exists i \in 2^I . \exists o \in 2^O . (\tau(s, i) = s') \wedge (l(s) = o) \wedge (q, i \cup o, q') \in \delta \ .$$

A run graph is accepting if every path (starting at the initial vertex (s_0, q_0)) has only finitely many visits of rejecting states. To show acceptance, we annotate every reachable node in the run graph with a natural number m, such that any path, starting in the initial state, contains less than m visits of rejecting states. Such an annotation exists if, and only if, the run graph is accepting [14].


$\top$
$\neg dec \wedge \neg int$
$\vee\ dec \wedge \neg val$
q_0
$\neg dec \wedge internal$
$\vee\ dec \wedge val$
q_1
q_2
pub
q_3
$(int \not\leftrightarrow res)$
int
q_e
$\neg int$
$\top$
$\neg pub_\pi \wedge \neg pub_{\pi'}$
$\wedge\ res_\pi \leftrightarrow res_{\pi'}$
q_0
$res_\pi \not\leftrightarrow res_{\pi'}$
q_e
$\top$

(a) Automaton accepting language defined by LTL formula in (2)

(b) Automaton accepting language defined by HyperLTL formula in (3)

Fig. 3. Universal co-Büchi automata recognizing the languages from Example 1.

Self-composition. The model checking of universal HyperLTL formulas [12] is based on self-composition. Let prj_i be the projection to the i-th element of a tuple. Let *zip* denote the usual function that maps a n-tuple of sequences to a single sequence of n-tuples, for example, $zip([1,2,3],[4,5,6]) = [(1,4),(2,5),(3,6)]$, and let *unzip* denote its inverse. The transition system $\mathcal{S}^n$ is the n-fold self-composition of $\mathcal{S} = \langle S, s_0, \tau, l\rangle$, if $\mathcal{S}^n = \langle S^n, s_0^n, \tau', l^n\rangle$ and for all $s, s' \in S^n$, $\alpha \in (2^I)^n$, and $\beta \in (2^O)^n$ we have that $\tau'(s,\alpha) = s'$ and $l^n(s) = \beta$ iff for all $1 \leq i \leq n$, it hold that $\tau(prj_i(s), prj_i(\alpha)) = prj_i(s')$ and $l(prj_i(s)) = prj_i(\beta)$. If T is the set of traces generated by $\mathcal{S}$, then $\{zip(t_1,\ldots,t_n) \mid t_1,\ldots,t_n \in T\}$ is the set of traces generated by $\mathcal{S}^n$.

We construct the universal co-Büchi automaton $\mathcal{A}_\psi$ such that the language of $\mathcal{A}_\psi$ is the set of words w such that $unzip(w) = \Pi$ and $\Pi \vDash_\emptyset \psi$, i.e., the tuple of traces that satisfy ψ. We get this automaton by dualizing the non-deterministic Büchi automaton for $\neg\psi$ [4], i.e., changing the branching from non-deterministic to universal and the acceptance condition from Büchi to co-Büchi. Hence, $\mathcal{S}$ satisfies a universal HyperLTL formula $\varphi = \forall\pi_1 \ldots \forall\pi_k.\, \psi$ if the traces generated by self-composition $\mathcal{S}^n$ are a subset of $\mathcal{L}(\mathcal{A}_\psi)$.

Lemma 3. *A transition system $\mathcal{S}$ satisfies the universal HyperLTL formula $\varphi = \forall\pi_1 \cdots \forall\pi_n.\, \psi$, if the run graph of $\mathcal{S}^n$ and $\mathcal{A}_\psi$ is accepting.*

Synthesis. Let $\mathcal{S} = \langle S, s_0, \tau, l\rangle$ and $\mathcal{A}_\psi = \langle Q, q_0, \delta, F\rangle$. We encode the synthesis problem as an SMT constraint system. Therefore, we use uninterpreted function symbols to encode the transition system and the annotation. For the transition system, those functions are the transition function $\tau : S \times 2^I \rightarrow S$ and the labeling function $l : S \rightarrow 2^O$. The annotation is split into two parts, a reachability constraint $\lambda^{\mathbb{B}} : S^n \times Q \rightarrow \mathbb{B}$ indicating whether a state in the run graph is reachable and a counter $\lambda^{\#} : S^n \times Q \rightarrow \mathbb{N}$ that maps every reachable vertex to the maximal number of rejecting states visited by any path starting in the initial vertex. The resulting constraint asserts that there is a transition system with accepting run graph.

$$\forall s, s' \in S^n.\, \forall q, q' \in Q.\, \forall i \in (2^I)^n.$$
$$\Big(\lambda^{\mathbb{B}}(s,q) \wedge \tau'(s,i) = s' \wedge (q, i \cup l(s), q') \in \delta\Big) \rightarrow \lambda^{\mathbb{B}}(s',q') \wedge \lambda^{\#}(s',q') \trianglerighteq \lambda^{\#}(s,q)$$

where $\trianglerighteq$ is $>$ if $q' \in F$ and $\geq$ otherwise.

Theorem 6. *The constraint system is satisfiable with bound b if, and only if, there is a transition system $\mathcal{S}$ of size b that realizes the HyperLTL formula.*

We extract a realizing implementation by asking the satisfiability solver to generate a model for the uninterpreted functions that encode the transition system.

5 Bounded Unrealizability

So far, we focused on the positive case, providing an algorithm for finding small solutions, if they exist. In this section, we shift to the case of detecting if a universal HyperLTL formula is unrealizable. We adapt the definition of counterexamples to realizability for LTL [15] to HyperLTL in the following. Let φ be a universal HyperLTL formula $\forall \pi_1 \cdots \forall \pi_n.\, \psi$ over inputs I and outputs O, a *counterexample to realizability* is a set of input traces $\mathcal{P} \subseteq (2^I)^\omega$ such that for every strategy $f : (2^I)^* \rightarrow 2^O$ the labeled traces $\mathcal{P}^f \subseteq (2^{I \cup O})^\omega$ satisfy $\neg\varphi = \exists \pi_1 \cdots \exists \pi_n.\, \neg\psi$.

Proposition 1. *A universal HyperLTL formula $\varphi = \forall \pi_1 \cdots \forall \pi_n.\, \psi$ is unrealizable if there is a counterexample $\mathcal{P}$ to realizability.*

Proof. For contradiction, we assume φ is realizable by a strategy f. As $\mathcal{P}$ is a counterexample to realizability, we know $\mathcal{P}^f \vDash \exists \pi_1 \cdots \exists \pi_n.\, \neg\psi$. This means that there exists an assignment $\Pi_{\mathcal{P}} \in \mathcal{V} \rightarrow \mathcal{P}^f$ with $\Pi_{\mathcal{P}} \vDash_{\mathcal{P}^f} \neg\psi$. Equivalently $\Pi_{\mathcal{P}} \nvDash_{\mathcal{P}^f} \psi$. Therefore, not all assignments $\Pi \in \mathcal{V} \rightarrow \mathcal{P}^f$ satisfy $\Pi \vDash_{\mathcal{P}^f} \psi$. Which implies $\mathcal{P}^f \nvDash \forall \pi_1 \cdots \forall \pi_n.\, \psi = \varphi$. Since φ is universal, we can defer $f \nvDash \varphi$, which concludes the contradiction. Thus, φ is unrealizable.

Despite being independent of strategy trees, there are in many cases finite representations of $\mathcal{P}$. Consider, for example, the unrealizable specification $\varphi_1 = \forall \pi \forall \pi'.\, \Diamond(i_\pi \leftrightarrow i_{\pi'})$, where the set $\mathcal{P}_1 = \{\emptyset^\omega, \{i\}^\omega\}$ is a counterexample to realizability. As a second example, consider $\varphi_2 = \forall \pi \forall \pi'.\, \Box(o_\pi \leftrightarrow o_{\pi'}) \wedge \Box(i_\pi \leftrightarrow \bigcirc o_\pi)$ with conflicting requirements on o. $\mathcal{P}_1$ is a counterexample to realizability for φ_2 as well: By choosing a different valuation of i in the first step, the system is forced to either react with different valuations of o (violating first conjunct), or not correctly repeating the initial value of i (violating second conjunct).

There are, however, already linear specifications where the set of counterexample paths is not finite and depends on the strategy tree [16]. For example, the specification $\forall \pi.\, \Diamond(i_\pi \leftrightarrow o_\pi)$ is unrealizable as the system cannot predict future values of the environment. There is no finite set of traces witnessing this: For every finite set of traces, there is a strategy tree such that $\Diamond(i_\pi \leftrightarrow o_\pi)$ holds on every such trace. On the other hand, there is a simple *counterexample strategy*, that is a strategy that observes output sequences and produces inputs. In this example, the counterexample strategy inverts the outputs given by the system, thus it is guaranteed that $\square(i \not\leftrightarrow o)$ for any system strategy.

We combine those two approaches, selecting counterexample paths and using strategic behavior. A k-counterexample strategy for HyperLTL observes k output sequences and produces k inputs, where k is a new parameter ($k \geq n$). The counterexample strategy is winning if (1) either the traces given by the system player do not correspond to a strategy, or (2) the body of the HyperLTL is violated for any n subset of the k traces. Regarding property (1), consider the two traces where the system player produces different outputs initially. Clearly, those two traces cannot be generated by any system strategy since the initial state (root labeling) is fixed.

The search for a k-counterexample strategy can be reduced to LTL synthesis using k-tuple input propositions O^k, k-tuple output propositions I^k, and the specification

$$\neg D_{I^k \mapsto O^k} \vee \bigvee_{P \subseteq \{1,\ldots,k\} \text{ with } |P|=n} \neg\psi[P] \enspace ,$$

where $\psi[P]$ denotes the replacement of a_{π_i} by the P_ith position of the combined input/output k-tuple.

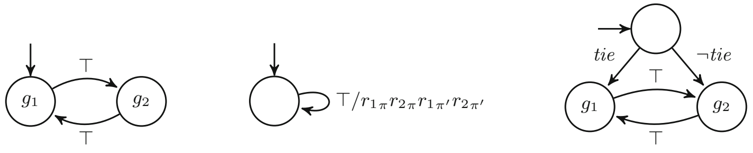
Theorem 7. *A universal HyperLTL formula $\varphi = \forall\pi_1 \cdots \forall\pi_n.\, \psi$ is unrealizable if there is a k-counterexample strategy for some $k \geq n$.*

6 Evaluation

We implemented a prototype synthesis tool, called BoSyHyper[1], for universal HyperLTL based on the bounded synthesis algorithm described in Sect. 4. Furthermore, we implemented the search for counterexamples proposed in Sect. 5. Thus, BoSyHyper is able to characterize realizability and unrealizability of universal HyperLTL formulas.

We base our implementation on the LTL synthesis tool BoSy [8]. For efficiency, we split the specifications into two parts, a part containing the linear (LTL) specification, and a part containing the hyperproperty given as HyperLTL formula. Consequently, we build two constraint systems, one using the standard bounded synthesis approach [14] and one using the approach described in Sect. 4. Before solving, those constraints are combined into a single SMT query. This results in a much more concise constraint system compared to the one where the complete specification is interpreted as a HyperLTL formula. For solving the SMT queries, we use the Z3 solver [20]. We continue by describing the benchmarks used in our experiments.

[1] BoSyHyper is available at https://www.react.uni-saarland.de/tools/bosy/.



(a) Non-symmetric solution (b) Counterexample to symmetry (c) Symmetry breaking solution

Fig. 4. Synthesized solution of the mutual exclusion protocols.

Symmetric Mutual Exclusion. Our first example demonstrates the ability to specify symmetry in HyperLTL for a simple mutual exclusion protocol. Let r_1 and r_2 be input signals representing mutual exclusive *requests* to a critical section and g_1/g_2 the respective grant to enter the section. Every request should be answered eventually $\square(r_i \rightarrow \diamondsuit g_i)$ for $i \in \{1, 2\}$, but not at the same time $\square\neg(g_1 \wedge g_2)$. The minimal LTL solution is depicted in Fig. 4a. It is well known that no mutex protocol can ensure perfect symmetry [19], thus when adding the symmetry constraint specified by the HyperLTL formula $\forall\pi\forall\pi'. (r_{1_\pi} \not\leftrightarrow r_{2_{\pi'}}) \mathcal{R} (g_{1_\pi} \leftrightarrow g_{2_{\pi'}})$ the formula becomes unrealizable. Our tool produces the counterexample shown in Fig. 4b. By adding another input signal *tie* that breaks the symmetry in case of simultaneous requests and modifying the symmetry constraint $\forall\pi\forall\pi'. ((r_{1_\pi} \not\leftrightarrow r_{2_{\pi'}}) \vee (tie_\pi \not\leftrightarrow \neg tie_{\pi'})) \mathcal{R} (g_{1_\pi} \leftrightarrow g_{2_{\pi'}})$ we obtain the solution depicted in Fig. 4c. We further evaluated the same properties on a version that forbids spurious grants, which are reported in Table 2 with prefix *full*.

Distributed and Fault-Tolerant Systems. In Sect. 3 we presented a reduction of arbitrary distributed architectures to HyperLTL. As an example for our evaluation, consider a setting with two processes, one for *encoding* input signals and one for *decoding*. Both processes can be synthesized simultaneously using a single HyperLTL specification. The (linear) correctness condition states that the decoded signal is always equal to the inputs given to the encoder. Furthermore, the encoder and decoder should solely depend on the inputs and the encoded signal, respectively. Additionally, we can specify desired properties about the encoding like fault-tolerance [16] or Hamming distance of code words [12]. The results are reported in Table 2 where i-j-x means i input bits, j encoded bits, and x represents the property. The property is either tolerance against a single Byzantine signal failure or a guaranteed Hamming distance of code words.

CAP Theorem. The CAP Theorem due to Brewer [2] states that it is impossible to design a distributed system that provides Consistency, Availability, and Partition tolerance (CAP) simultaneously. This example has been considered before [16] to evaluate a technique that could automatically detect unrealizability. However, when we drop either Consistency, Availability, or Partition tolerance, the corresponding instances (AP, CP, and CA) become realizable, which the previous work was not able to prove. We show that our implementation can show both, unrealizability of CAP and realizability of AP, CP, and CA. In contrast to the previous encoding [16] we are not limited to acyclic architectures.

Long-term Information-flow. Previous work on model-checking hyperproperties [12] found that an implementation for the commonly used *I2C* bus protocol could remember input values ad infinitum. For example, it could not be verified that information given to the implementation eventually leaves it, i.e., is forgotten. This is especially unfortunate in high security contexts. We consider a simple bus protocol which is inspired by the widely used *I2C* protocol. Our example protocol has the inputs *send* for initiating a transmission, *in* for the value that should be transferred, and an *ack*nowledgment bit indicating successful transmission. The bus master waits in an *idle* state until a *send* is received. Afterwards, it transmits a header sequence, followed by the value of *in*, waits for an acknowledgement and then indicates *success* or *failure* to the sender before returning to the idle state. We specify the property that the *in*put has no influence on the *data* that is send, which is obviously violated (instance NI1). As a second property, we check that this information leak cannot happen arbitrary long (NI2) for which there is a realizing implementation.

Dining Cryptographers. Recap the dining cryptographers problem introduced earlier. This benchmark is interesting as it contains two types of hyperproperties. First, there is information-flow between the three cryptographers, where some secrets (s_{ab}, s_{ac}, s_{bc}) are shared between pairs of cryptographers. In the formalization, we have 4 entities: three processes describing the 3 cryptographers (out_i) and one process computing the result (p_g), i.e., whether the group has paid or not, from out_i. Second, the final result should only disclose whether one of the cryptographers has paid or the NSA. This can be formalized as a indistinguishability property between different executions. For example, when we compare the two traces π and π' where C_a has paid on π and C_b has paid on π'. Then the outputs of both have to be the same, if their common secret s_{ab} is different on those two traces (while all other secrets s_{ac} and s_{bc} are the same). This ensures that from an outside observer, a flipped output can be either result of a different shared secret or due to the announcement. Lastly, the linear specification asserts that $p_g \leftrightarrow \neg p_{NSA}$.

Results. Table 2 reports on the results of the benchmarks. We distinguish between state-labeled (*Moore*) and transition-labeled (*Mealy*) transition systems. Note that the counterexample strategies use the opposite transition system, i.e., a Mealy system strategy corresponds to a state-labeled (Moore) environment strategy. Typically, Mealy strategies are more compact, i.e., need smaller transition systems and this is confirmed by our experiments. BoSyHyper is able to solve most of the examples, providing realizing implementations or counterexamples. Regrading the unrealizable benchmarks we observe that usually two simultaneously generated paths $(k = 2)$ are enough with the exception

Table 2. Results of BoSyHyper on the benchmarks sets described in Sect. 6. They ran on a machine with a dual-core Core i7, 3.3 GHz, and 16 GB memory.

Benchmark	Instance	Result	States		Time[sec.]	
			Moore	Mealy	Moore	Mealy
Symmetric Mutex	non-sym	realizable	2	2	1.4	1.3
	sym	unrealizable ($k = 2$)	1	1	1.9	2.0
	tie	realizable	3	3	1.7	1.6
	full-non-sym	realizable	4	4	1.4	1.4
	full-sym	unrealizable ($k = 2$)	1	1	4.3	6.2
	full-tie	realizable	9	5	1 802.7	5.2
Encoder/Decoder	1-2-hamming-2	realizable	4	1	1.6	1.3
	1-2-fault-tolerant	unrealizable ($k = 2$)	1	-	54.9	-
	1-3-fault-tolerant	realizable	4	1	151.7	1.7
	2-2-hamming-2	unrealizable ($k = 3$)	-	1	-	10.6
	2-3-hamming-2	realizable	16	1	>1 h	1.5
	2-3-hamming-3	unrealizable ($k = 3$)	-	1	-	126.7
CAP Theorem	cap-2-linear	realizable	8	1	7.0	1.3
	cap-2	unrealizable ($k = 2$)	1	-	1 823.9	-
	ca-2	realizable	-	1	-	4.4
	ca-3	realizable	-	1	-	15.0
	cp-2	realizable	1	1	1.8	1.6
	cp-3	realizable	1	1	3.2	10.6
	ap-2	realizable	-	1	-	2.0
	ap-3	realizable	-	1	-	43.4
Bus Protocol	NI1	unrealizable ($k = 2$)	1	1	75.2	69.6
	NI2	realizable	8	8	24.1	33.9
Dining Cryptographers	secrecy	realizable	-	1	-	82.4

of the encoder example. Overall the results are encouraging showing that we can solve a variety of instances with non-trivial information-flow.

7 Conclusion

In this paper, we have considered the reactive realizability problem for specifications given in the temporal logic HyperLTL. We gave a complete characterization of the decidable fragments based on the quantifier prefix and, additionally, identified a decidable fragment in the, in general undecidable, universal fragment of HyperLTL. Furthermore, we presented two algorithms to detect realizable and unrealizable HyperLTL specifications, one based on bounding the system implementation and one based on bounding the number of counterexample paths. Our prototype implementation shows that our approach is able to synthesize systems with complex information-flow properties.

References

1. Agrawal, S., Bonakdarpour, B.: Runtime verification of k-safety hyperproperties in HyperLTL. In: Proceedings of CSF, pp. 239–252. IEEE Computer Society (2016)
2. Brewer, E.A.: Towards robust distributed systems (abstract). In: Proceedings of ACM, p. 7. ACM (2000)
3. Chaum, D.: Security without identification: transaction systems to make big brother obsolete. Commun. ACM **28**(10), 1030–1044 (1985)
4. Clarkson, M.R., Finkbeiner, B., Koleini, M., Micinski, K.K., Rabe, M.N., Sánchez, C.: Temporal logics for hyperproperties. In: Abadi, M., Kremer, S. (eds.) POST 2014. LNCS, vol. 8414, pp. 265–284. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54792-8_15
5. Clarkson, M.R., Schneider, F.B.: Hyperproperties. J. Comput. Secur. **18**(6), 1157–1210 (2010)
6. Dimitrova, R., Finkbeiner, B., Kovács, M., Rabe, M.N., Seidl, H.: Model checking information flow in reactive systems. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI 2012. LNCS, vol. 7148, pp. 169–185. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27940-9_12
7. Faymonville, P., Finkbeiner, B., Rabe, M.N., Tentrup, L.: Encodings of bounded synthesis. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 354–370. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_20
8. Faymonville, P., Finkbeiner, B., Tentrup, L.: BoSy: an experimentation framework for bounded synthesis. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 325–332. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_17
9. Finkbeiner, B., Hahn, C.: Deciding hyperproperties. In: Proceedings of CONCUR. LIPIcs, vol. 59, pp. 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016)
10. Finkbeiner, B., Hahn, C., Stenger, M., Tentrup, L.: Monitoring hyperproperties. In: Lahiri, S., Reger, G. (eds.) RV 2017. LNCS, vol. 10548, pp. 190–207. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67531-2_12
11. Finkbeiner, B., Jacobs, S.: Lazy synthesis. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI 2012. LNCS, vol. 7148, pp. 219–234. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27940-9_15
12. Finkbeiner, B., Rabe, M.N., Sánchez, C.: Algorithms for model checking HyperLTL and HyperCTL*. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 30–48. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_3
13. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: Proceedings of LICS, pp. 321–330. IEEE Computer Society (2005)
14. Finkbeiner, B., Schewe, S.: Bounded synthesis. STTT **15**(5–6), 519–539 (2013)
15. Finkbeiner, B., Tentrup, L.: Detecting unrealizable specifications of distributed systems. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 78–92. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_6
16. Finkbeiner, B., Tentrup, L.: Detecting unrealizability of distributed fault-tolerant systems. Log. Methods Comput. Sci. **11**(3) (2015)
17. Finkbeiner, B., Zimmermann, M.: The first-order logic of hyperproperties. In: Proceedings of STACS. LIPIcs, vol. 66, pp. 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017)

18. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: Proceedings of FOCS, pp. 531–542. IEEE Computer Society (2005)
19. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems - Safety. Springer, New York (1995). https://doi.org/10.1007/978-1-4612-4222-2
20. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
21. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Proceedings of FOCS, pp. 746–757. IEEE Computer Society (1990)
22. Post, E.L.: A variant of a recursively unsolvable problem. Bull. Am. Math. Soc. **52**(4), 264–268 (1946)