




Realizing ω -regular Hyperproperties

Bernd Finkbeiner , Christopher Hahn , Jana Hofmann  ,
and Leander Tentrup 

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



Abstract. We study the expressiveness and reactive synthesis problem of HyperQPTL, a logic that specifies ω -regular hyperproperties. HyperQPTL is an extension of linear-time temporal logic (LTL) with explicit trace and propositional quantification and therefore *truly* combines trace relations and ω -regularity. As such, HyperQPTL can express promptness, which states that there is a common bound on the number of steps up to which an event must have happened. We demonstrate how the HyperQPTL formulation of promptness differs from the type of promptness expressible in the logic Prompt-LTL. Furthermore, we study the realizability problem of HyperQPTL by identifying decidable fragments, where one decidable fragment contains formulas for promptness. We show that, in contrast to the satisfiability problem of HyperQPTL, propositional quantification has an immediate impact on the decidability of the realizability problem. We present a reduction to the realizability problem of HyperLTL, which immediately yields a bounded synthesis procedure. We implemented the synthesis procedure for HyperQPTL in the bounded synthesis tool BoSy. Our experimental results show that a range of arbiter satisfying promptness can be synthesized.

1 Introduction

Hyperproperties [5], which are mainly studied in the area of secure information flow control, are a generalization from trace properties to *sets* of trace properties. That is, they relate multiple execution traces with each other. Examples are noninterference [20], observational determinism [34], symmetry [16], or promptness [24], i.e., properties whose satisfaction cannot be determined by analyzing each execution trace in isolation.

A number of logics have been introduced to express hyperproperties (examples are [4, 19, 25]). They either add explicit trace quantification to a temporal logic or build on monadic first-order or second-order logics and add an equal-level predicate, which connects traces with each other. A comprehensive study comparing such hyperlogics has been initiated in [6].

This work was partially supported by the Collaborative Research Center “Foundations of Perspicuous Software Systems” (TRR 248, 389792660) and by the European Research Council (ERC) Grant OSARES (No. 683300).

© The Author(s) 2020

S. K. Lahiri and C. Wang (Eds.): CAV 2020, LNCS 12225, pp. 40–63, 2020.

https://doi.org/10.1007/978-3-030-53291-8_4

The most prominent hyperlogic is HyperLTL [4], which extends classic linear-time temporal logic (LTL) [26] with trace variables and explicit trace quantification. HyperLTL has been successfully applied in (runtime) verification, (e.g., [15, 21, 32]), specification analysis [11, 14], synthesis [12, 13], and program repair [1] of hyperproperties. As an example specification, the following HyperLTL formula expresses observational determinism by stating that for every pair of traces, if the observable inputs I are the same on both traces, then also the observable outputs O have to agree

$$\forall\pi\forall\pi'. \Box(I_\pi = I_{\pi'}) \rightarrow \Box(O_\pi = O_{\pi'}) . \quad (1)$$

Thus, hyperlogics can not only specify functional correctness, but may also enforce the absence of information leaks or presence of information propagation. There is a great practical interest in information flow control, which makes synthesizing implementations that satisfy hyperproperties highly desirable. Recently [12], it was shown that the synthesis problem of HyperLTL, although undecidable in general, remains decidable for many fragments, such as the $\exists^*\forall$ fragment. Furthermore, a *bounded synthesis* procedure was developed, for which a prototype implementation based on BoSy [7, 9, 12] showed promising results.

HyperLTL is, however, intrinsically limited in expressiveness. For example, promptness is not expressible in HyperLTL. Promptness is a property stating that there is a bound b , common for all traces, on the number of steps up to which an event e must have happened. Additionally, just like LTL, HyperLTL can express neither ω -regular nor epistemic properties [2, 29]. Epistemic properties are statements about the transfer of knowledge between several components. An exemplary epistemic specification is described by the *dining cryptographers problem* [3]: three cryptographers sit at a table in a restaurant. Either one of the cryptographers or, alternatively, the NSA must pay for their meal. The question is whether there is a protocol where each cryptographer can find out whether the NSA or one of the cryptographers paid the bill, without revealing the identity of the paying cryptographer.

In this paper, we explore HyperQPTL [6, 29], a hyperlogic that is more expressive than HyperLTL. Specifically, we study its expressiveness and reactive synthesis problem. HyperQPTL extends HyperLTL with quantification over sequences of new propositions. What makes the logic particularly expressive is the fact that the trace quantifiers and propositional quantifiers can be freely interleaved. With this mechanism, HyperQPTL can not only express all ω -regular properties over a sequences of n -tuples; it truly interweaves trace quantification and ω -regularity. For example, promptness can be stated as the following HyperQPTL formula:

$$\exists b. \forall\pi. \Diamond b \wedge (\neg b \mathcal{U} e_\pi) . \quad (2)$$

The formula states that there exists a sequence $s \in (2^{\{a\}})^\omega$, such that event e holds on all traces before the first occurrence of b in s . In this paper, we argue that the type of promptness expressible in HyperQPTL is incomparable to the expressiveness of Prompt-LTL [24], a logic introduced to express promptness

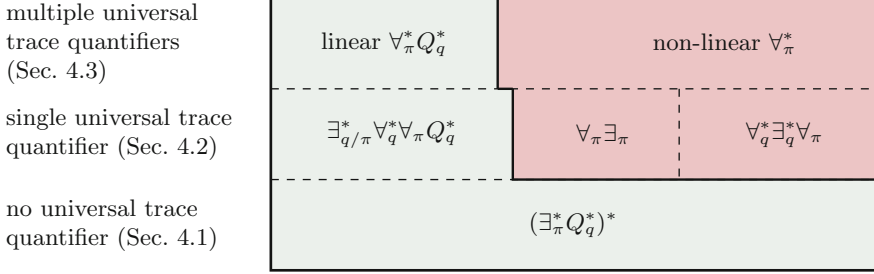


Fig. 1. The realizability problem of HyperQPTL. Left and below of the solid line are the decidable fragments, right above the solid line the undecidable fragments.

properties. It is further known that HyperQPTL also subsumes epistemic extensions of temporal logics such as $LTL_{\mathcal{K}}$ [22], as well as the first-order hyperlogic $FO[\langle, E]$ [6, 19, 29]. Its expressiveness makes HyperQPTL particularly interesting. The model checking problem of HyperQPTL is, despite the logic being quite expressive, decidable [29]. We also explore an alternative definition of HyperQPTL that would result in an even more expressive logic. However, we show that the logic would have an undecidable model checking problem, which constitutes a major drawback in the context of computer-aided verification. Furthermore, satisfiability is decidable for large fragments of the logic [6]. Decidable HyperQPTL fragments can be described solely in terms of their *trace* quantifier prefix. This indicates that propositional quantification has no negative impact on the decidability, although it greatly increases the expressiveness. We establish that propositional quantification, in contrast to the satisfiability problem, has an impact on the realizability problem: it becomes undecidable when combining a propositional $\forall \exists$ quantifier alternation with a single universal trace quantifier. However, we show that the synthesis problem of large HyperQPTL fragments remains decidable, where one of these fragments contains promptness properties. We partially obtain these results by reducing the HyperQPTL realizability problem to the HyperLTL realizability problem. Based on this reduction, we extended the BoSy bounded synthesis tool to also synthesize systems respecting HyperQPTL specifications. We provide promising experimental results of our prototype implementation: using BoSy and HyperQPTL specifications, we were able to synthesize arbiters that respect promptness.

This paper is structured as follows. In Sect. 2, we give necessary preliminaries. In Sect. 3, we define HyperQPTL. We discuss an alternative approach to define a logic expressing ω -regular hyperproperties, before pointing out that its model checking problem is undecidable. Subsequently, we give examples for the expressiveness of HyperQPTL, namely by characterizing the type of promptness properties HyperQPTL can express. Additionally, we recapitulate how HyperQPTL also subsumes epistemic properties. Section 4 discusses the realizability problem of HyperQPTL. We describe HyperQPTL fragments in terms of their quantifier prefixes. To present our results, we use the following notation. We write \forall_{π}

and \forall_q for a single universal trace and propositional quantifier, respectively. To denote a sequence of universal trace and propositional quantifiers, we write \forall_π^* and \forall_q^* . Furthermore, we use $\forall_{\pi/q}^*$ for a sequence of mixed universal quantification. We use the analogous notation for existential quantifiers. Lastly, Q_π^* and Q_q^* denote a sequence of mixed universal and existential trace and propositional quantifiers, respectively. As an example, the $\forall_\pi^* Q_q^*$ fragment denotes all formulas of the form $\forall \pi_1 \dots \forall \pi_m. \exists / \forall q_1 \dots \exists / \forall q_n. \varphi$, where φ is quantifier free. Figure 1 summarizes our results. We establish that a major factor for the decidability of the realizability problem consists in the number of universal trace occurring in a formula. Realizability of HyperQPTL formulas without \forall_π quantifiers is decidable (Sect. 4.1). Formulas with a single \forall_π are decidable if they belong to the $\exists_{q/\pi}^* \forall_q^* \forall_\pi Q_q^*$ fragment. This fragment also contains promptness. For more than one universal trace quantifier, we show that decidability can be guaranteed for a fragment that we call the linear $\forall_\pi^* Q_q^*$ fragment. We also show that all the above fragments are tight, i.e., realizability of all other formulas is in general undecidable. Lastly, Sect. 5 presents experiments for the prototype implementation of our bounded synthesis algorithm for HyperQPTL.

2 Preliminaries

We use AP for a set of atomic propositions. A *trace* over AP is an infinite sequence $t \in (2^{\text{AP}})^\omega$. For $i \in \mathbb{N}$, we write $t[i]$ for the i th element of t and $t[i, \infty]$ for the suffix of t starting from position i . For two traces t, t' over AP and a set $\text{AP}' \subseteq \text{AP}$, we write $t =_{\text{AP}'} t'$ to indicate that t and t' agree on all $a \in \text{AP}'$, and respectively $T =_{\text{AP}'} T'$ for two sets of traces T and T' . Furthermore, we define a replacement function $t[q \mapsto t_q]$ that given a trace t and a trace $t_q \in (2^{\{q\}})^\omega$, replaces the occurrences of q in t according to t_q , such that $t[q \mapsto t_q] =_{\{q\}} t_q$ and $t[q \mapsto t_q] =_{\text{AP} \setminus \{q\}} t$. We also lift this notation to sets of traces and define $T[q \mapsto t_q] = \{t[q \mapsto t_q] \mid t \in T\}$.

QPTL [31] extends Linear Temporal Logic (LTL) with quantification over propositions. QPTL formulas φ are defined as follows.

$$\begin{aligned} \varphi &::= \exists q. \varphi \mid \forall q. \varphi \mid \psi \\ \psi &::= q \mid \neg \psi \mid \psi \vee \psi \mid \bigcirc \psi \mid \diamond \psi \end{aligned}$$

where $q \in \text{AP}$ and AP is a set of atomic propositions. For simplicity, we assume that variable names in formulas are cleared of double occurrences. The semantics of φ over AP is defined with respect to a trace $t \in (2^{\text{AP}})^\omega$.

$t \models q$	iff	$q \in t[0]$
$t \models \neg\psi$	iff	$t \not\models \psi$
$t \models \psi_1 \vee \psi_2$	iff	$t \models \psi_1$ or $t \models \psi_2$
$t \models \bigcirc\psi$	iff	$t[1, \infty] \models \psi$
$t \models \diamond\psi$	iff	$\exists i \geq 0. t[i, \infty] \models \psi$
$t \models \exists q. \varphi$	iff	$\exists t_q \in (2^{\{q\}})^\omega. t[q \mapsto t_q] \models \varphi$
$t \models \forall q. \varphi$	iff	$\forall t_q \in (2^{\{q\}})^\omega. t[q \mapsto t_q] \models \varphi$

We did not define the until operator \mathcal{U} as native part of the logic. It can be derived using propositional quantification [23]. The boolean connectives $\wedge, \rightarrow, \leftrightarrow$ and the temporal operators globally \square and release \mathcal{R} are derived as usually.

3 ω -Regular Hyperproperties

Just like LTL, HyperLTL cannot express ω -regular languages [29]. LTL can be extended to QPTL by adding quantification over atomic propositions. In QPTL, ω -regular languages become expressible. We therefore study HyperQPTL [6, 29], the extension of HyperLTL with propositional quantification, to express ω -regular hyperproperties. Given a set AP of atomic propositions and a set \mathcal{V} of trace variables, the syntax of HyperQPTL is defined as follows

$$\begin{aligned} \varphi &::= \forall\pi. \varphi \mid \exists\pi. \varphi \mid \forall q. \varphi \mid \exists q. \varphi \mid \psi \\ \psi &::= a_\pi \mid q \mid \neg\psi \mid \psi \vee \psi \mid \bigcirc\psi \mid \diamond\psi, \end{aligned}$$

where $a, q \in \text{AP}$ and $\pi \in \mathcal{V}$. As for QPTL, we assume that formulas are cleared of double occurrences of variable names. We require that in well-defined HyperQPTL formulas, each a_π is in the scope of a trace quantifier binding π and each q is in the scope of a propositional quantifier binding q . Note that atomic propositions a_π refer to a quantified trace π , whereas quantified propositional variables q are independent of the traces. The semantics of a well-defined HyperQPTL formula over AP is defined with respect to a set of traces $T \subseteq (2^{\text{AP}})^\omega$ and an assignment function $\Pi : \mathcal{V} \rightarrow T$. We define the satisfaction relation $\Pi, i \models_T \varphi$ as follows:

$\Pi, i \models_T a_\pi$	iff	$a \in \Pi(\pi)[i]$
$\Pi, i \models_T q$	iff	$\forall t \in T. q \in t[i]$
$\Pi, i \models_T \neg\psi$	iff	$\Pi, i \not\models_T \psi$
$\Pi, i \models_T \psi_1 \vee \psi_2$	iff	$\Pi, i \models_T \psi_1 \vee \Pi, i \models_T \psi_2$
$\Pi, i \models_T \bigcirc\psi$	iff	$\Pi, i + 1 \models_T \psi$
$\Pi, i \models_T \diamond\psi$	iff	$\exists j \geq i. \Pi, j \models_T \psi$
$\Pi, i \models_T \exists\pi. \varphi$	iff	$\exists t \in T. \Pi[\pi \mapsto t], i \models_T \varphi$
$\Pi, i \models_T \forall\pi. \varphi$	iff	$\forall t \in T. \Pi[\pi \mapsto t], i \models_T \varphi$
$\Pi, i \models_T \exists q. \varphi$	iff	$\exists t_q \in (2^{\{q\}})^\omega. \Pi, i \models_{T[q \mapsto t_q]} \varphi$
$\Pi, i \models_T \forall q. \varphi$	iff	$\forall t_q \in (2^{\{q\}})^\omega. \Pi, i \models_{T[q \mapsto t_q]} \varphi$

Note that the semantics of propositional quantification is defined in such a way that in the scope of a quantifier binding q , all traces agree on their q -sequence. We say that a set of traces T satisfies a HyperQPTL formula φ if $\emptyset, 0 \models_T \varphi$, where \emptyset is the empty trace assignment. QPTL formulas can be expressed in HyperQPTL using a single universal trace quantifier. Furthermore, HyperLTL [4] is the syntactic subset of HyperQPTL that does not contain propositional quantification.

While HyperQPTL can express a wide range of properties (see Sect. 3.1), its model checking problem is still decidable [29]. Furthermore, the syntactic fragments for which satisfiability is decidable can be expressed solely in terms of the occurring trace quantifiers: Just like for HyperLTL, satisfiability of a HyperQPTL formula is decidable if no $\forall\pi$ is followed by an $\exists\pi$ [6].

The definition of HyperQPTL is straightforward, however, one could argue that it is not the only way to extend QPTL to a hyperlogic. The original idea of QPTL is to “color” the trace by introducing additional atomic propositions. The way HyperQPTL is defined, that idea is translated to sets of traces by coloring the traces uniformly. An alternative approach could be to color every trace individually by introducing a full atomic proposition for every propositional quantification. This resembles full second-order quantification and would therefore result in a considerably more expressive logic. In particular, we show that the model checking problem would become undecidable, which is, especially in the context of automatic verification, unfavorable. For the remainder of this section, we call the logic resulting from the alternative definition HyperQPTL⁺. The syntax of HyperQPTL⁺ is similar to the one of HyperQPTL, just without the rule q for the evaluation of the propositional variables. This accounts for the idea that the propositional quantification can freely reassign atomic propositions; thus, there is no need to distinguish between free atomic propositions and quantified atomic propositions:

$$\begin{aligned} \varphi &::= \forall\pi. \varphi \mid \exists\pi. \varphi \mid \forall a. \varphi \mid \exists a. \varphi \mid \psi \\ \psi &::= a_\pi \mid \neg\psi \mid \psi \vee \psi \mid \bigcirc\psi \mid \diamond\psi \end{aligned}$$

Semantically, only the rules for the quantification of the propositional quantifiers change:

$$\begin{aligned} \Pi, i \models_T \exists a. \varphi & \quad \text{iff} & \quad \exists T' \subseteq (2^{\text{AP}})^\omega. T' =_{\text{AP} \setminus \{a\}} T \wedge \Pi, i \models_{T'} \varphi \\ \Pi, i \models_T \forall a. \varphi & \quad \text{iff} & \quad \forall T' \subseteq (2^{\text{AP}})^\omega. T' =_{\text{AP} \setminus \{a\}} T \rightarrow \Pi, i \models_{T'} \varphi . \end{aligned}$$

Lemma 1. *The HyperQPTL⁺ model checking problem is undecidable.*

Proof. Given a finite Kripke structure K and a HyperQPTL⁺ formula φ , the model checking problem asks whether the trace set T produced by K satisfies φ . The proof follows the undecidability proof for the model checking problem of S1S[E] [6], a logic which lifts S1S to the level of hyperlogics. We describe a reduction from the halting problem of 2-counter machines (which are Turing complete) to the HyperQPTL⁺ model checking problem. A 2-counter machine (2CM) consists of a finite set of serially numbered instructions that modify two counters. A configuration of a 2CM is a triple $(n, v_1, v_2) \in \mathbb{N}^3$, where n determines the next instruction to be executed, and v_1 and v_2 assign the counter values. Each instruction can either increase or decrease one of the counters; or test either of the counters for zero and, depending on the outcome, jump to another instruction. Furthermore, we assume a special instruction i_{halt} , which indicates that the machine has reached a halting state. A 2CM halts from initial configuration s_0 if there is a finite sequence s_0, \dots, s_n of configurations such that s_n is a halting configuration and s_{i+1} is a result of applying the instruction in s_i to configuration s_i . Let \mathcal{M} be a 2CM. We describe T and φ such that $T \models \varphi$ iff \mathcal{M} halts. We choose $\text{AP} = \{i, c_1, c_2\}$ and T is the set of all traces where each atomic proposition holds exactly once. That way, a trace t encodes a configuration of the machine: If $i \in t[n]$, $c_1 \in t[v_1]$, and $c_2 \in t[v_2]$, the machine is in configuration (n, v_1, v_2) . It is easy to see that T can be produced by a finite Kripke structure. To describe φ , we make two helpful observations. First, using propositional quantification, we can quantify a trace set $T_q \subseteq T$: a trace t is in T_q iff the quantified proposition q eventually occurs on t . Second, for two traces $t, t' \in T$, we can state that t' encodes a configuration which is the successor of the configuration encoded by t . Using these observations, we define $\varphi = \exists q. \varphi'$, where q encodes a set $T_q \subseteq T$ that is supposed to describe a halting computation. To ensure that T_q describes a halting computation, φ' is a conjunction of the following requirements: T_q must

1. be finite,
2. contain a halting configuration and the initial configuration,
3. be predecessor closed with respect to the encoded configurations it contains (except for the initial configuration).

Finiteness of T_q can be expressed by stating that there is an upper bound on the values of i, c_1 , and c_2 on the traces in T_q . With the observations made before, stating the above requirements in HyperQPTL⁺ now remains a straightforward exercise. \square

Since the model checking problem of HyperQPTL⁺ is undecidable, we focus on HyperQPTL to express ω -regular hyperproperties. In particular, we show that HyperQPTL can express a range of relevant properties that are neither expressible in HyperLTL, nor in QPTL.

3.1 The Expressiveness of HyperQPTL

HyperQPTL combines trace quantification with ω -regularity. The interplay between the two features enables HyperQPTL to express a variety of properties. In Sect. 1, we showed how HyperQPTL can express a form of promptness. In this section, we further elaborate on the type of properties HyperQPTL can express. In particular, we compare it to Prompt-LTL, a logic that extends LTL with bounded eventualities. Furthermore, HyperQPTL is also able to express epistemic properties by emulating the knowledge operator known from LTL_K.

A straightforward class of properties HyperQPTL can express are ω -regular properties over n -tuples of quantified traces. Formulas expressing this type of properties first have a trace quantifier prefix followed by a QPTL formula, i.e., they lie in the $Q_\pi^* Q_q^*$ fragment. This fragment of HyperQPTL corresponds to the extension of QPTL with *prenex* trace quantification. However, the true expressive power of HyperQPTL originates from the fact that we allow the trace quantifiers and propositional quantifiers to alternate.

Promptness Properties. Promptness properties are an example for HyperQPTL's interplay between trace quantification and propositional quantification. Promptness expresses that eventualities are fulfilled within a bounded number of steps. One way to express promptness properties is the logic Prompt-LTL, which extends LTL with the promptness operator \Diamond_p . A system satisfies a Prompt-LTL formula φ if there is a bound k such that all traces of the system fulfill the formula where each \Diamond_p in φ is replaced by $\Diamond^{\leq k}$, i.e., the system must fulfill all prompt eventualities within k steps. For example, $\varphi = \Box \Diamond_p \psi$ holds in a system if there is a bound k such that all traces of the system at all times satisfy ψ within k steps. HyperQPTL can express a different type of promptness properties. In Sect. 1, Formula 2, we showed how one can state in HyperQPTL that there is a bound, common for all traces, until which an eventuality has to be fulfilled. The idea is to quantify a new proposition b , such that the first position in which b is true serves as the bound. Compared to Prompt-LTL, HyperQPTL thus expresses a weaker form of promptness, while still being stronger than pure eventuality. This type of promptness only becomes meaningful when comparing several traces of the system: HyperQPTL can enforce that there is a common bound for all traces (the system cannot starve), but it does not make the bound explicit. The following example shows a more involved promptness property expressible in HyperQPTL.

Example 1. HyperQPTL can express *bounded waiting for a grant*. It states that if the system requests access to a shared resource at point in time t , then it will be granted access within a bounded amount of time. The bound may depend on

the point in time t where access to the resource was requested. However, it may not depend on the current trace. We express this property in HyperQPTL as follows, also adding that the system will not request access twice without being granted access in between.

$$\forall \pi. \Box (r_\pi \rightarrow \bigcirc (\neg r_\pi \mathcal{W} g_\pi)) \quad (1)$$

$$\forall \pi. \exists b. \forall \pi'. \Box (r_\pi \wedge r_{\pi'} \rightarrow \bigcirc (\Diamond b \wedge (\neg b \mathcal{U} g_\pi) \wedge (\neg b \mathcal{U} g_{\pi'}))) \quad (2)$$

Formula 1 states that no second request is posed before being given a grant. Formula 2 expresses the bounded waiting property by universally quantifying a trace, then existentially quantifying a sequence of bounds b . Now, for every trace π' , whenever π and π' pose a request at the same point in time, both have to get access to the resource before b holds next. Therefore, for each point in time, there is a bound such that all traces posing a request at that point in time get access within a bounded number of steps. Note that this property differs from saying “all traces are eventually granted access”, where the bound may also depend on the trace under consideration. In this scenario, each of the infinitely many traces could wait arbitrarily long for the grant. In particular, it could happen that with each trace the waiting time is longer than before.

The above example shows how the interplay of trace quantifiers and propositional quantifiers can be leveraged to express a new class of promptness properties. We finally note that compared to Prompt-LTL, HyperQPTL cannot express that all eventualities must be fulfilled within a fixed k number of steps.

Corollary 1. *The expressiveness of HyperQPTL and Prompt-LTL is incomparable.*

Epistemic Properties. Another interesting class of properties that are not expressible in HyperLTL are epistemic properties. Epistemic properties describe the knowledge of agents that interact with each other in a system. Logics that express epistemic properties are often equipped with a so-called knowledge operator, e.g., $\text{LTL}_{\mathcal{K}}$, which is LTL extended with the knowledge operator $\mathcal{K}_A \varphi$. The operator denotes that an agent $A \subseteq \text{AP}$ knows φ . An agent A is characterized in terms of the atomic propositions he can observe. The semantics of the operator is described with the following rule

$$t, i \models \mathcal{K}_A \varphi \quad \text{iff} \quad \forall t'. t[0, i] =_A t'[0, i] \rightarrow t', i \models \varphi .$$

The formula is evaluated with respect to a trace t and a position i . We omit the semantic definition for the rest of the logic, which corresponds to plain LTL. The semantic definition of the operator captures the idea that an agent knows some fact φ if φ holds on all traces that are indistinguishable for the agent.

Example 2 (Dining Cryptographers). The dining cryptographers problem [3] is an interesting example of how epistemic properties can characterize non-trivial

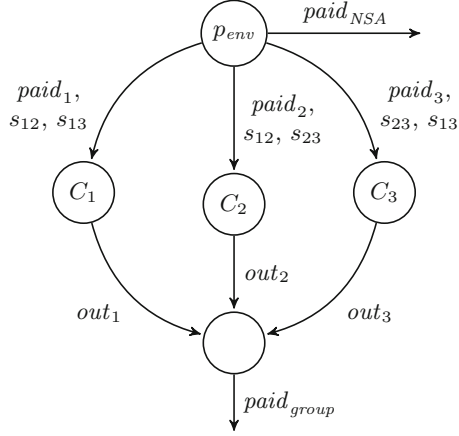


Fig. 2. The dining cryptographers problem with three cryptographers.

protocols. The problem describes the following situation (see Fig. 2): three cryptographers C_1 , C_2 , and C_3 sit at a table in a restaurant and either one of cryptographers or, alternatively, the NSA paid for their meal. The task for the cryptographers is to figure out whether the NSA or one of the cryptographers paid. However, if one of the cryptographers paid, then the others must not be able to infer who it was. Each cryptographer C_i receives several bits of information: $paid_i$ indicating whether or not he pays the bill, and two secrets, each shared with one of the other cryptographers. The secrets can be used to encode the information they share as output out_i . By combining the outputs of all cryptographers, it must become clear whether the NSA or one of the group paid. The specification of the protocol can be easily formalized in $LTL_{\mathcal{K}}$. The following formula describes the desired behavior of agent C_1 :

$$\begin{aligned}
 DC_{agent1} := & \\
 & (paid_{group} \wedge \neg paid_1 \rightarrow (\mathcal{K}_{C_1}(paid_2 \vee paid_3) \wedge \neg \mathcal{K}_{C_1} paid_2 \wedge \neg \mathcal{K}_{C_1} paid_3)) \\
 & \wedge (paid_{NSA} \rightarrow \mathcal{K}_{C_1}(\neg paid_1 \wedge \neg paid_2 \wedge \neg paid_3)) .
 \end{aligned}$$

The knowledge operator can also be defined for hyperlogics [29]. It receives an additional parameter π , indicating the trace the knowledge refers to. When added to HyperQPTL, it has the following semantics:

$$\Pi, i \models_T \mathcal{K}_{A, \pi} \varphi \quad \text{iff} \quad \forall t' \in T. \Pi(\pi)[0, i] =_A t'[0, i] \rightarrow \Pi[\pi \mapsto t'], i \models_T \varphi .$$

The knowledge operator, however, can be encoded in HyperQPTL using propositional quantification. Epistemic problems, such as the dining cryptographers problem, can thus be expressed in HyperQPTL.

Theorem 1 ([29]). *HyperQPTL can emulate the knowledge operator.*

Proof. We recap the proof from [29]: Let $\varphi = Q_{\pi/q} \dots Q_{\pi/q} \cdot \varphi'$ be a HyperQPTL formula, equipped with the knowledge operator as defined above. We assume that φ is given in negated normal form, i.e. each $\mathcal{K}_{A,\pi}$ occurs either in positive position or in negated form. Let u and t be fresh propositions and let π' be a fresh trace variable. Recursively, we replace each knowledge operator $\mathcal{K}_{A,\pi}$ occurring in φ in positive position with the following formula

$$Q_{\pi/q} \dots Q_{\pi/q} \cdot \exists u. \forall r. \forall \pi'. \varphi'[\mathcal{K}_{A,\pi} \psi \mapsto u] \wedge ((r \mathcal{U} (u \wedge r \wedge \bigcirc \square \neg r)) \wedge \square(r \rightarrow A_\pi = A_{\pi'})) \rightarrow \square(r \wedge \bigcirc \neg r \rightarrow \psi[\pi \mapsto \pi'])$$

and each $\mathcal{K}_{A,\pi}$ occurring negatively with the following formula

$$Q_{\pi/q} \dots Q_{\pi/q} \cdot \exists u. \forall r. \exists \pi'. \varphi'[\neg \mathcal{K}_{A,\pi} \psi \mapsto u] \wedge ((r \mathcal{U} (u \wedge r \wedge \bigcirc \square \neg r)) \rightarrow \square(r \rightarrow A_\pi = A_{\pi'}) \wedge \square(r \wedge \bigcirc \neg r \rightarrow \neg \psi[\pi \mapsto \pi'])),$$

where we use $\varphi'[\mathcal{K}_{A,\pi} \psi \mapsto u]$ to denote that in φ' , a *single* occurrence of the knowledge operator is replaced by u , and $\psi[\pi \mapsto \pi']$ to denote the formula where π is replaced by π' . The existentially quantified proposition u indicates the points in time where the knowledge operator is supposed to hold/not hold. The universally quantified proposition r is assumed to change once from r to $\neg r$ and thereby point at one of the points in time picked by u . It is then used to compare the prefix of the old trace π and an alternative trace quantified by the trace variable π' . \square

4 HyperQPTL Realizability

In reactive synthesis, the task is, given a specification φ , to construct a system that satisfies the specification. More precisely, the system is assumed to receive some inputs from an environment and has to react with outputs such that the specification is fulfilled. The realizability problem asks for the existence of a so-called *strategy tree*, where the edges are labeled with all possible inputs and the task is to find a function f that labels the nodes with the corresponding outputs. Figure 3 shows a strategy tree for a single input bit i . We define strategies following [12]. Let a set $AP = I \dot{\cup} O$ be given. A *strategy* $f: (2^I)^* \rightarrow 2^O$ maps sequences of input valuations 2^I to an output valuation 2^O . For an infinite word $w = w_0 w_1 w_2 \dots \in (2^I)^\omega$, the trace corresponding to a strategy f is defined as $(f(\epsilon) \cup w_0)(f(w_0) \cup w_1)(f(w_0 w_1) \cup w_2) \dots \in (2^{I \dot{\cup} O})^\omega$. For any trace $w = w_0 w_1 w_2 \dots \in (2^{I \dot{\cup} O})^\omega$ and strategy $f: (2^I)^* \rightarrow 2^O$, we lift the set containment operator \in defining that $w \in f$ iff $f(\epsilon) = w_0 \cap O$ and $f((w_0 \cap I) \dots (w_i \cap I)) = w_{i+1} \cap O$ for all $i \geq 0$. We say that a strategy f satisfies a HyperQPTL formula φ over $AP = I \dot{\cup} O$ iff $\{w \mid w \in f\}$ satisfies φ .

With the definition of a strategy at hand, we can define the realizability problem of HyperQPTL formally.

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

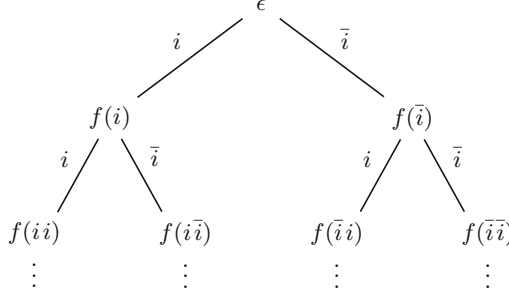


Fig. 3. A strategy tree for the reactive realizability problem.

For technical reasons, we assume (without loss of generality) that quantified atomic propositions are classified as outputs, not inputs. This complies with the intuition that propositional quantifiers should be a means for additional expressiveness; they should not overwrite the inputs received from the environment. The definition of realizability of QPTL and HyperLTL specifications is inherited from the definition for HyperQPTL.

Compared to the standard realizability problem, the distributed realizability problem is defined over an architecture, containing a number of processes interacting with each other. The goal is to find a strategy for each of the processes. In the following proofs, we will make use of the distributed realizability problem of QPTL, which we therefore also define formally.

A *distributed architecture* [17, 27] A over atomic propositions AP is a tuple $\langle P, p_{env}, \mathcal{I}, \mathcal{O} \rangle$, where P is a finite set of processes and $p_{env} \in P$ is a designated environment process. The functions $\mathcal{I} : P \rightarrow 2^{AP}$ and $\mathcal{O} : P \rightarrow 2^{AP}$ define the inputs and outputs of processes. The output of one process can be the input of another process. The output of the processes must be pairwise disjoint, i.e., for all $p \neq p' \in P$ it holds that $\mathcal{O}(p) \cap \mathcal{O}(p') = \emptyset$. We assume that the environment process forwards inputs to the processes and has no input of its own, i.e., $\mathcal{I}(p_{env}) = \emptyset$.

Definition 2 (Distributed QPTL Realizability [17]). A QPTL formula φ over free atomic propositions AP is *realizable in an architecture* $A = \langle P, p_{env}, \mathcal{I}, \mathcal{O} \rangle$ if for each process $p \in P$, there is a strategy $f_p : (2^{\mathcal{I}(p)})^* \rightarrow 2^{\mathcal{O}(p)}$ such that the combination of all f_p satisfies φ .

The distributed realizability problem for QPTL is (inherited from LTL) in general undecidable [27]. However, we will use the result that the problem remains decidable for architectures without *information forks* [17]. The notion of information forks captures the flow of data in the system. Intuitively, an architecture contains an information fork if the processes cannot be ordered linearly according to their informedness. Formally, an information fork in an architecture $A = \langle P, p_{env}, \mathcal{I}, \mathcal{O} \rangle$ is defined as a tuple (P', V', p, p') , where p, p' are two different processes, $P' \subseteq P$, and $V' \subseteq AP$ is disjoint from $\mathcal{I}(p) \cup \mathcal{I}(p')$. (P', V', p, p')



(a) Information fork: An architecture with two processes; process p produces output o from input i and p' produces output o' from input i' . (b) No information fork: The same architecture as on the left, where the inputs of process p' are changed to i and i' .

Fig. 4. Distributed architectures

is an information fork if P' together with the edges that are labeled with at least one variable from V' forms a subgraph rooted in the environment and there exist two nodes $q, q' \in P'$ that have edges to p, p' , respectively, such that $\mathcal{O}(q) \cap \mathcal{I}(p) \not\subseteq \mathcal{I}(p')$ and $\mathcal{O}(q') \cap \mathcal{I}(p') \not\subseteq \mathcal{I}(p)$. The definition formalizes the intuition that p and p' receive incomparable input bits, i.e., they have incomparable information.

Example 3. Two example architectures are depicted in Fig. 4 [12]. The processes in Fig. 4a receive distinct inputs and thus neither process is more informed than the other. The architecture therefore contains an information fork with $P' = \{env, p, p'\}$, $V' = \{i, i'\}$, $q = env$, $q' = env$. The processes in Fig. 4b can be ordered linearly according to the subset relation on the inputs and thus the architecture contains no information fork.

In the following sections, we identify tight syntactic fragments of HyperQPTL for which the standard realizability problem is decidable. We give decidability proofs and show that formulas outside the decidable fragments are in general undecidable. An important aspect for decidability is the number of universal trace quantifiers that appear in the formula. We thus present our findings in three categories, depending on the number of universal trace quantifiers a formula has.

4.1 No Universal Trace Quantifier

We show that the realizability problem of any HyperQPTL formula without a \forall_π quantifier is decidable. The problem is reduced to QPTL realizability.

Theorem 2. *Realizability of the $(\exists_\pi^* Q_q^*)^*$ fragment of HyperQPTL is decidable.*

Proof. Let a $(\exists_\pi^* Q_q^*)^*$ HyperQPTL formula φ over $AP = I \dot{\cup} O = \{a^0, \dots, a^k\}$ with trace quantifiers π_0, \dots, π_n be given. We reduce the problem to the realizability problem of QPTL, which is known to be decidable (since QPTL formulas can be translated to Büchi automata). The idea is to replace each existential trace quantifier $\exists \pi_i$ with quantification of propositions $a_{\pi_i}^0, a_{\pi_i}^1, \dots, a_{\pi_i}^k$, one for

each $a^j \in \text{AP}$, thereby mimicking the quantification of a trace. To make sure that only traces from an actual strategy tree are chosen, we add a dependency formula which forces the outputs to be dependent on the inputs. The following QPTL formula implements the idea.

$$\varphi_{QPTL} := \varphi [i \leq n : \exists \pi_i \mapsto \exists a_{\pi_i}^0 \dots \exists a_{\pi_i}^k.] \wedge \bigwedge_{i \leq n} \bigwedge_{j \leq n} (I_{\pi_i} \neq I_{\pi_j}) \mathcal{R}(O_{\pi_i} = O_{\pi_j})$$

We use the notation $[i \leq n : \exists \pi_i \mapsto \exists a_{\pi_i}^0 \dots \exists a_{\pi_i}^k.]$ to indicate that each π_i for $0 \leq i \leq n$ is replaced with the respective series of existential propositional quantification. Furthermore, we write $I_{\pi_i} \neq I_{\pi_j}$ as syntactic sugar for $\bigvee_{a \in I} a_{\pi_i} \leftrightarrow a_{\pi_j}$ (and similarly for $O_{\pi_i} = O_{\pi_j}$). We show that φ and φ_{QPTL} are equirealizable. For the first direction, assume that φ is realizable by a strategy f . Notice that all atomic propositions in φ_{QPTL} are bound by a propositional quantifier. Therefore, if the witness sequences for the quantified propositions can be chosen correctly, any strategy realizes φ_{QPTL} . Propositions $a_{\pi_i}^j$ are chosen according to the witness traces of $f \models \varphi$. Witnesses for the remaining atomic propositions are also chosen according to their witnesses from $f \models \varphi$. Now, the first conjunct of φ_{QPTL} is fulfilled since $f \models \varphi$ holds. The second conjunct is fulfilled since any two traces π_i, π_j of a strategy tree fulfill by construction $(I_{\pi_i} \neq I_{\pi_j}) \mathcal{R}(O_{\pi_i} = O_{\pi_j})$. For the other direction, assume that φ_{QPTL} is realizable (by construction independently from the strategy). Let $t_{a_{\pi_0}^0}, \dots, t_{a_{\pi_n}^k}$ be the witness sequences for the respective quantified atomic propositions. The following strategy realizes φ .

$$f(\sigma) = \begin{cases} \{t_{a_{\pi_i}}[|\sigma|] \mid a \in O\} & \text{if for some } i \leq n, \\ \sigma = \{t_{a_{\pi_i}}[0] \mid a \in I\} \dots \{t_{a_{\pi_i}}[|\sigma|] \mid a \in I\} \\ \emptyset & \text{otherwise} \end{cases}$$

Strategy f chooses the outputs according to the witnesses for the propositions encoding the traces. Note that because of the second conjunct in φ_{QPTL} , the output is always unique, even if several encoded traces start with the same input sequence. Now, $f \models \varphi$ holds because of the first conjunct of φ_{QPTL} . \square

4.2 Single Universal Trace Quantifier

In this fragment, we allow exactly one universal trace quantifier. It is particularly interesting as it contains many promptness properties. For example, the following promptness formulation mentioned in the introduction lies within the fragment:

$$\exists b. \forall \pi. \diamond b \wedge (\neg b \mathcal{U} e_\pi) .$$

Theorem 3. *Realizability of the $\exists_{q/\pi}^* \forall_{\pi}^* \forall_{q} Q_q^*$ fragment is decidable.*

We show the theorem in two steps. First, we generalize a proof from [12], showing that realizability of the $\exists_{\pi}^* \forall_{\pi} Q_q^*$ fragment is decidable. Second, we show that we can reduce the realizability problem of any HyperQPTL formula to a formula where some propositional quantifiers are replaced with trace quantifiers.

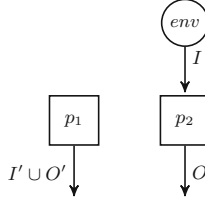


Fig. 5. Distributed architecture encoding existential choice of traces.

Lemma 2. *Realizability of the $\exists_{\pi}^* \forall_{\pi} Q_q^*$ fragment is decidable.*

Proof. The reasoning generalizes the proof in [12] showing that realizability $\exists_{\pi}^* \forall_{\pi}$ HyperLTL formulas is decidable. We reduce the problem to the distributed realizability problem of QPTL without information forks, which is—since QPTL is subsumed by the μ -calculus—decidable [17]. Let a HyperQPTL formula $\varphi = \exists \pi_1 \dots \exists \pi_n. \forall \pi. \psi$ over $AP = I \dot{\cup} O$ be given, where ψ is from the Q_q^* fragment. We define a distributed architecture \mathcal{A} over an extended set of atomic propositions $AP' = I \cup O \cup I' \cup O'$. Similarly to the proof in Theorem 2, I' and O' are composed of a copy of the atomic propositions for each existentially quantified variable π_j . Formally, $I' = \bigcup_{1 \leq j \leq n} \{i_{\pi_j} \mid i \in I\}$ and $O' = \bigcup_{1 \leq j \leq n} \{o_{\pi_j} \mid o \in O\}$. Now we define \mathcal{A} as follows.

$$\begin{aligned} \mathcal{A} &:= \langle (p_{env}, p_1, p_2), p_{env}, \mathcal{I}, \mathcal{O}, \rangle \\ \mathcal{I} &:= (p_1 \mapsto \emptyset, p_2 \mapsto I) \\ \mathcal{O} &:= (p_{env} \mapsto I, p_1 \mapsto I' \cup O', p_2 \mapsto O) \end{aligned}$$

The architecture is displayed in Fig. 5. The idea is that process p_1 sets the values of all i_{π_j} and o_{π_j} (for $j \leq n$) and thereby determines the choice for the existentially quantified traces. Process p_1 receives no input and therefore needs to make a deterministic choice. Process p_2 then solves the realizability of formula $\forall \pi. \psi$. The following QPTL formula φ' encodes the idea.

$$\varphi' := \psi' \wedge \left(\bigwedge_{1 \leq j \leq n} (I_{\pi_j} \neq I) \mathcal{R}(O_{\pi_j} = O) \right),$$

where ψ' is defined as ψ , where all a_{π} are replaced by a (but atomic propositions a_{π_j} are still part of ψ'). Note that QPTL formulas implicitly quantify over all traces universally. Similarly to the proof in Theorem 2, the second conjunct ensures that process p_1 encodes actual paths from the strategy tree of process p_2 (which is also the strategy tree for formula φ). Thus, φ' is realizable for the distributed architecture \mathcal{A} iff φ is realizable. \square

To state the second lemma, we need to define what it means to replace quantifiers in a formula. Let $\varphi = Q_{\pi/q_1} \dots Q_{\pi/q_n} \psi$ be a HyperQPTL formula, and J be a set of indices such that for all $j \in J$, there exists a propositional quantifier

$\exists q_j$ or $\forall q_j$ in φ . Furthermore, assume that no π_j with $j \in J$ occurs in φ and that $a \in \text{AP}$. We denote by $\varphi[J \hookrightarrow_a \pi]$ the formula where each propositional quantifier $\exists q_j$ (or $\forall q_j$, respectively) with $j \in J$ is replaced with the corresponding trace quantifier $\exists \pi_j$ (or $\forall \pi_j$, respectively); and each q_j in ψ is replaced by a_{π_j} .

Lemma 3. *Let any HyperQPTL formula φ over $\text{AP} = I \dot{\cup} O$ and a set of indices J be given. If $\varphi[J \hookrightarrow_i \pi]$ is realizable, then so is φ , where $i \in I$ is an arbitrary input, assuming w.l.o.g., that I is non-empty.*

Proof. Let φ and J be given. Formula $\varphi[J \hookrightarrow_i \pi]$ replaces the quantification over sequences $(2^{\{q\}})^\omega$ with trace quantification, where the trace is only used for statements about a single input i . We thus exploit the fact that in the realizability problem, there is a trace for every input sequence. Therefore, the transformed formula is equirealizable. \square

Now, we have everything we need to prove Theorem 3.

Proof (of Theorem 3). Let φ be a HyperQPTL formula of the $\exists_{q/\pi}^* \forall_q^* \forall_\pi Q_q^*$ fragment. First, observe that in the quantifier prefix of φ , the \forall_q^* quantifiers and the \forall_π can be swapped. The resulting formula belongs to the $\exists_{q/\pi}^* \forall_\pi Q_q^*$ fragment. By Lemma 3, the formula can be transformed to a equirealizable formula of the $\exists_\pi \forall_\pi Q_q^*$ fragment, for which realizability is decidable by Lemma 2. \square

Lemma 3 allows us to decide realizability of a HyperQPTL formula by replacing propositional quantifiers with trace quantifiers. Thus, we can reduce HyperQPTL realizability to HyperLTL realizability, a fact that we use in Sect. 5 to describe a bounded synthesis algorithm for HyperQPTL.

Corollary 2. *The realizability problem of HyperQPTL can be soundly reduced to the realizability problem of HyperLTL.*

Lastly, we show that the decidable fragment is tight in the class of formulas with a single universal trace quantifier. We do so by showing that a propositional $\forall_q^* \exists_q^*$ quantifier alternation followed by a single trace quantifier \forall_π leads to an undecidable realizability problem. The proof is carried out by a reduction from Post's Correspondence Problem.

Theorem 4. *Realizability is undecidable for HyperQPTL formulas with a single \forall_π quantifier outside the $\exists_{q/\pi}^* \forall_q^* \forall_\pi Q_q^*$ fragment.*

Proof. Inherited from HyperLTL, realizability of formulas with a \forall_π quantifier followed by an \exists_π quantifier is undecidable [12]. It remains to show that realizability of formulas from the $\forall_q^* \exists_q^* \forall_\pi$ fragment is in general undecidable. We give a reduction from Post's Correspondence Problem (PCP) [28] to a HyperQPTL formula from the $\forall_q^* \exists_q^* \forall_\pi$ fragment. In PCP, we are given two equally long lists α and β consisting of finite words from some alphabet Σ of size n . PCP is the problem to find an index sequence $(i_k)_{1 \leq k \leq K}$ with $K \geq 1$ and $1 \leq i_k \leq n$, such that $\alpha_{i_1} \dots \alpha_{i_K} = \beta_{i_1} \dots \beta_{i_K}$. Intuitively, PCP is the problem of choosing an infinite sequence of domino stones (with finitely many different stones), where each stone

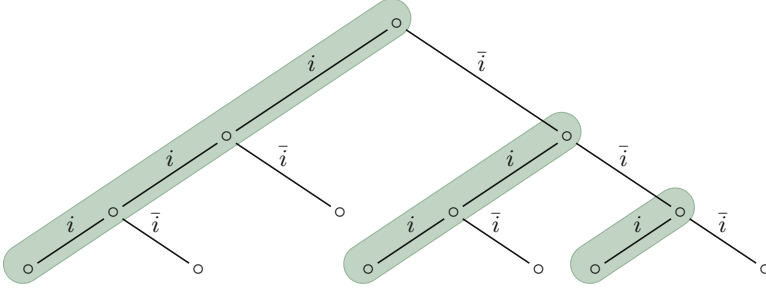


Fig. 6. A sketch of the strategy tree of our PCP reduction: relevant traces are marked in green. (Color figure online)

consists of two words α_i and β_i . Let a PCP instance with $\Sigma = \{a_1, a_2, \dots, a_n\}$ and two lists α and β be given. We choose our set of atomic propositions as follows: $\text{AP} := I \dot{\cup} O$ with $I := \{i\}$ and $O := (\Sigma \cup \{\dot{a}_1, \dot{a}_2, \dots, \dot{a}_n\} \cup \#)^2$, where we use the dot symbol to encode that a stone starts at this position of the trace. We write \tilde{a} to denote either a or \dot{a} . The single input i spans a binary strategy tree. We encode the PCP instance into a HyperQPTL formula that is realizable if and only if the PCP instance has a solution:

$$\forall q_i. \forall \mathbf{q}. \exists p_i. \exists \mathbf{p}. \forall \pi. ((\Box \pi = p_i) \rightarrow (\Box \pi = \mathbf{p})) \wedge \\ ((\Box \pi = (q_i, \mathbf{q})) \rightarrow \varphi_{\text{reduc}}(q_i, \mathbf{q}, p_i, \mathbf{p})) ,$$

where \mathbf{q} and \mathbf{p} are sequences of universally and existentially quantified propositional variables, such that for each $(o, o') \in O$, there is a $q_{(o, o')} \in \mathbf{q}$ and a $p_{(o, o')} \in \mathbf{p}$. Together with q_i and p_i for the input i , they simulate a universally and an existentially quantified trace from the model. The notation $\pi = \mathbf{q}$ denotes that for every $q_a \in \mathbf{q}$, it holds that $a_\pi \leftrightarrow q_a$. As seen before, the premise $(\Box \pi = (q_i, \mathbf{q}))$ and the conjunct $(\Box \pi = p_i) \rightarrow (\Box \pi = \mathbf{p})$ ensure that the propositions (q_i, \mathbf{q}) and (p_i, \mathbf{p}) are chosen to represent actual traces from the model. The universal quantification π thus only ensures that (q_i, \mathbf{q}) and (p_i, \mathbf{p}) , which are used for the main reduction, are chosen correctly. The reduction is implemented in the formula φ_{reduc} and follows the construction in [10], where it is shown that the satisfiability and realizability problem of HyperLTL are undecidable for a $\forall \exists$ trace quantifier prefix.

$$\varphi_{\text{reduc}}(q_i, \mathbf{q}, p_i, \mathbf{p}) := \varphi_{\text{rel}}(q_i) \rightarrow \varphi_{\text{is}++}(q_i, p_i) \\ \wedge \varphi_{\text{start}}(\varphi_{\text{stone\&shift}}(\mathbf{q}, \mathbf{p}), q_i) \wedge \varphi_{\text{sol}}(q_i, \mathbf{q})$$

- $\varphi_{\text{rel}}(q_i) := \neg q_i \mathcal{U} \Box q_i$ defines the set of *relevant* traces through the binary strategy tree (see Fig. 6).
- $\varphi_{\text{is}++}(q_i, p_i) := (\neg q_i \wedge \neg p_i) \mathcal{U} (\Box q_i \wedge \neg p_i \wedge \bigcirc \Box p_i)$ defines that a relevant trace is the direct successor trace of another relevant trace.

- $\varphi_{\text{sol}}(q_i, \mathbf{q}) := \Box q_i \rightarrow ((\bigvee_{i=1}^n q_{(\dot{a}_i, \dot{a}_i)}) \wedge (\bigvee_{i=1}^n q_{(\dot{a}_i, \bar{a}_i)})) \mathcal{U} \Box q_{(\#, \#)}$ ensures that the path on which globally i holds is a “solution” trace, i.e., encodes the PCP solution sequence.
- $\varphi_{\text{start}}(\varphi, q_i) := \neg q_i \mathcal{U} (\varphi \wedge \Box q_i)$ cuts off an irrelevant prefix until φ starts.
- $\varphi_{\text{stone\&shift}}(\mathbf{q}, \mathbf{p})$ encodes that the trace simulated by \mathbf{q} starts with a valid encoding of a stone from the PCP instance and that the trace simulated by \mathbf{p} encodes the same trace but with the first stone removed (see [10]).

For example, let α with $\alpha_1 = a$, $\alpha_2 = ab$, $\alpha_3 = bba$, and β with $\beta_1 = baa$, $\beta_2 = aa$ and $\beta_3 = bb$ be given. A possible solution for this PCP instance is $(3, 2, 3, 1)$, since $bbaabbbbaa = i_\alpha = i_\beta$. The full sequence at the trace $\Box i$ represents the solution with the outputs

$$(\dot{b}, \dot{b})(b, b)(a, \dot{a})(\dot{a}, a)(b, \dot{b})(\dot{b}, b)(b, \dot{b})(a, a)(\dot{a}, a)(\#, \#)(\#, \#) \dots$$

The next relevant trace, therefore, contains

$$(\dot{a}, \dot{a})(b, a)(\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \dots$$

Continuing this, the following relevant traces are:

$$\begin{aligned} &(\dot{b}, \dot{b})(b, b)(a, \dot{b})(\dot{a}, a)(\#, a)(\#, \#)(\#, \#) \dots \\ &(\dot{a}, \dot{b})(\#, a)(\#, a)(\#, \#)(\#, \#) \dots \\ &(\#, \#)(\#, \#) \dots \end{aligned}$$

The relevant traces verify the solution provided on the $\Box i$ trace by removing one stone after the other. Thus, the formula is realizable iff the PCP instance has a solution. \square

4.3 Multiple Universal Trace Quantifiers

When considering multiple universal trace quantifiers \forall_π^* , the problem becomes undecidable. This is because in HyperLTL, one can encode distributed architectures – for which the problem is undecidable – directly into the formula without using any propositional quantification [12].

Corollary 3. *Realizability of the \forall_π^* fragment is in general undecidable.*

However, we show that the realizability problem for formulas with more than one universal trace quantifier is decidable if we restrict ourselves to formulas in the so-called *linear fragment*, i.e., that does not allow an encoding of a distributed architecture. We define the linear fragment of HyperQPTL, where the definitions are adopted from [12].

Let $A, C \subseteq \text{AP}$. We define that atomic propositions $c \in C$ do solely depend on propositions $a \in A$ as the HyperQPTL formula

$$D_{A \rightarrow C} := \forall \pi \forall \pi'. \left(\bigvee_{a \in A} (a_\pi \leftrightarrow a_{\pi'}) \right) \mathcal{R} \left(\bigwedge_{c \in C} (c_\pi \leftrightarrow c_{\pi'}) \right).$$

We define a *collapse* function, which collapses a HyperQPTL formula with a \forall_π^* universal quantifier prefix into a formula with a single \forall_π quantifier. Propositional quantifiers are preserved by the operation. Let φ be $\forall\pi_1 \dots \forall\pi_n. Q_q^*. \psi$. We define the collapsed formula of φ as $\text{collapse}(\varphi) := \forall\pi. Q_q^*. \psi[\pi_1 \mapsto \pi][\pi_2 \mapsto \pi] \dots [\pi_n \mapsto \pi]$ where $\psi[\pi_i \mapsto \pi]$ replaces all occurrences of π_i in ψ with π .

Lemma 4. *Either $\varphi \equiv \text{collapse}(\varphi)$ or φ has no equivalent $\forall_\pi^*. Q_q^*$ formula.*

Proof. The collapse function solely works on the trace quantification mechanism of the HyperQPTL formula, by reducing them to a single universal quantification. The theorem has been proven for \forall^* HyperLTL formulas in [12]. Inner propositional quantification does not interfere with this mechanism, hence, the proof can be carried out identically. \square

Now we can formally define the linear \forall_π^* fragment. Intuitively, we require that every input-output dependency can be ordered linearly, i.e., we are restricted to linear architectures without information forks (see Example 3).

Definition 3. *Let $O = \{o_1, \dots, o_n\}$. A HyperQPTL formula φ is called linear if for all $o_i \in O$ there is a $J_i \subseteq I$ such that $\varphi \wedge D_{I \mapsto O} \equiv \text{collapse}(\varphi) \wedge \bigwedge_{o_i \in O} D_{J_i \mapsto \{o_i\}}$ and $J_i \subseteq J_{i+1}$ for all $i \leq n$.*

This results in the following corollary. Since the universal quantifiers can be collapsed, the resulting problem is the realizability problem of QPTL in a linear architecture, which is decidable [17].

Corollary 4. *Realizability of the linear $\forall_\pi^* Q_q^*$ fragment is decidable.*

Remark on Complexities. Our aim was to work out the largest possible fragments for which the realizability problem of HyperQPTL remains decidable. The three fragments for which we could prove decidability all subsume the logic QPTL, for which the realizability problem is known to be non-elementary (already its satisfiability problem is non-elementary [30]). Hence, realizability of the discussed HyperQPTL fragments has a non-elementary lower bound. Finding interesting fragments for which the problem has a more feasible complexity therefore remains an open challenge.

5 Experiments

We have implemented a prototype tool that can solve the HyperQPTL realizability problem using the bounded synthesis approach [18]. More concretely, we extended the HyperLTL synthesis tool BoSy [7, 9, 12]. BoSy reduces the HyperLTL synthesis problem to a SMT constraint system which is then solved by z3 [8] (for more see [12]). We implemented the reduction of HyperQPTL synthesis to HyperLTL synthesis (Corollary 2) in BoSy, such that the tool can also handle HyperQPTL formulas. We evaluated the tool against a range of

Table 1. Experimental results for prompt arbiter

Instance	Bound on system	Bound on \exists -strategy	Result	Time [sec.]
arbiter-2-prompt	2	1	unsat	<1
	2	2	sat	<1
arbiter-2-full-prompt	3	1	unsat	2.4
	3	2	sat	6.0
arbiter-3-prompt	3	1	unsat	4.2
	3	2	sat	9.5
arbiter-4-prompt	4	1	unsat	97
	4	2	?	TO

benchmarks sets, shown in Table 1. The first column indicates the parameterized benchmark name. The second and third columns indicate the bounds given to the bounded synthesis procedure. The second column is the bound on the size of the system. The newest version of BoSy also bounds the size of the strategy for the existential player, this bound is given in column three. For a detailed explanation of how existential strategies are bounded in BoSy, we refer to [7].

We synthesized a range of resource arbiters. Our benchmark set is parametric in the number of clients that can request access to the shared resource (written arbiter- k -prompt where k is the number of clients in Table 1). Unlike normal arbiters, we require the arbiter to fulfill promptness for some of the clients, i.e., requests must be answered within a bounded number of steps [33]. We state the promptness requirement in HyperQPTL by applying the *alternating-color technique* from [24]. Intuitively, the alternating-color technique works as follows: We quantify a q -sequence that “changes color” between q and $\neg q$. Each change of color is used as a potential bound. Once a request occurs, the grant must be given within two changes of color. Thus, the HyperQPTL formulation amounts to the following specifications, here exemplary for 2 clients, where we require promptness only for client 1.

$$\forall \pi. \Box \neg (g_\pi^1 \wedge g_\pi^2) \quad (1)$$

$$\forall \pi. \Box (r_\pi^2 \rightarrow \Diamond g_\pi^2) \quad (2)$$

$$\exists q. \forall \pi. \Box \Diamond q \wedge \Box \Diamond \neg q \quad (3)$$

$$\begin{aligned} & \wedge \Box (r_\pi^1 \rightarrow (q \rightarrow (q \mathcal{U} (\neg q \mathcal{U} g_\pi^1)))) \\ & \quad \wedge (\neg q \rightarrow (\neg q \mathcal{U} (q \mathcal{U} g_\pi^1))) \\ & \forall \pi. (\neg g_\pi^1 \mathcal{W} r_\pi^1) \wedge (\neg g_\pi^2 \mathcal{W} r_\pi^2) \end{aligned} \quad (4)$$

Formula 1 states mutual exclusion. Formula 2 states that client 2 must be served eventually (but not within a bounded number of steps). Formula 3 states the promptness requirement for client 1. It quantifies an alternating q -sequence, which serves as a sequence of global bounds that must be respected on all traces π . Then, if client 1 poses a request, the grant must be given within two changes of the value of q . Formula 4 is only added in benchmarks named *arbiter- k -full-prompt*. It specifies that no spurious grants should be given.

BoSy successfully synthesizes prompt arbiter of up to 3 states. For a 4-state prompt arbiter BoSy did not return in reasonable time.

6 Conclusion

We studied the hyperlogic HyperQPTL, which combines the concepts of trace relations and ω -regularity. We showed that HyperQPTL is very expressive, it can express properties like *promptness*, *bounded waiting for a grant*, *epistemic* properties, and, in particular, any ω -regular property. Those properties are not expressible in previously studied hyperlogics like HyperLTL. At the same time, we argued that the expressiveness of HyperQPTL is optimal in a sense that a more expressive logic for ω -regular hyperproperties would have an undecidable model checking problem. We furthermore studied the realizability problem of HyperQPTL. We showed that realizability is decidable for HyperQPTL fragments that contain properties like promptness. But still, in contrast to the satisfiability problem, propositional quantification does make the realizability problem of hyperlogics harder. More specifically, the HyperQPTL fragment of formulas with a universal-existential propositional quantifier alternation followed by a single trace quantifier is undecidable in general, even though the projection of the fragment to HyperLTL has a decidable realizability problem. Lastly, we implemented the bounded synthesis problem for HyperQPTL in the prototype tool BoSy. Using BoSy with HyperQPTL specifications, we have been able to synthesize several resource arbiters. The synthesis problem of non-linear-time hyperlogics is still open. For example, it is not yet known how to synthesize systems from specifications given in branching-time hyperlogics like HyperCTL*.

References

1. Bonakdarpour, B., Finkbeiner, B.: Program repair for hyperproperties. In: Chen, Y.-F., Cheng, C.-H., Esparza, J. (eds.) ATVA 2019. LNCS, vol. 11781, pp. 423–441. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31784-3_25
2. Bozzelli, L., Maubert, B., Pinchinat, S.: Unifying hyper and epistemic temporal logics. In: Pitts, A. (ed.) FoSSaCS 2015. LNCS, vol. 9034, pp. 167–182. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46678-0_11
3. Chaum, D.: Security without identification: transaction systems to make big brother obsolete. *Commun. ACM* **28**(10), 1030–1044 (1985). <https://doi.org/10.1145/4372.4373>

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). <https://doi.org/10.3233/JCS-2009-0393>
6. Coenen, N., Finkbeiner, B., Hahn, C., Hofmann, J.: The hierarchy of hyperlogics. In: 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2019), pp. 1–13 (2019). <https://doi.org/10.1109/LICS.2019.8785713>
7. Coenen, N., Finkbeiner, B., Sánchez, C., Tentrup, L.: Verifying hyperliveness. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 121–139. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_7
8. 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
9. Faymonville, P., Finkbeiner, B., Tentrup, L.: BoSy: an experimentation framework for bounded synthesis. In: Majumdar, R., Kunčák, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 325–332. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_17
10. Finkbeiner, B., Hahn, C.: Deciding hyperproperties. In: Proceedings of CONCUR. LIPIcs, vol. 59, pp. 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016). <https://doi.org/10.4230/LIPIcs.CONCUR.2016.13>
11. Finkbeiner, B., Hahn, C., Hans, T.: MGHYPHER: checking satisfiability of hyperltl formulas beyond the $\exists^*\forall^*$ fragment. In: Lahiri, S.K., Wang, C. (eds.) ATVA 2018. LNCS, vol. 11138, pp. 521–527. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4_31
12. Finkbeiner, B., Hahn, C., Lukert, P., Stenger, M., Tentrup, L.: Synthesizing reactive systems from hyperproperties. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 289–306. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_16
13. Finkbeiner, B., Hahn, C., Lukert, P., Stenger, M., Tentrup, L.: Synthesis from hyperproperties. *Acta Inf.* **57**(1), 137–163 (2020). <https://doi.org/10.1007/s00236-019-00358-2>
14. Finkbeiner, B., Hahn, C., Stenger, M.: EAHyper: satisfiability, implication, and equivalence checking of hyperproperties. In: Majumdar, R., Kunčák, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 564–570. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_29
15. Finkbeiner, B., Hahn, C., Torfah, H.: Model checking quantitative hyperproperties. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 144–163. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_8
16. 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
17. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: Proceedings of LICS, pp. 321–330. IEEE Computer Society (2005). <https://doi.org/10.1109/LICS.2005.53>
18. Finkbeiner, B., Schewe, S.: Bounded synthesis. *STTT* **15**(5–6), 519–539 (2013). <https://doi.org/10.1007/s10009-012-0228-z>

19. 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). <https://doi.org/10.4230/LIPIcs.STACS.2017.30>
20. Goguen, J.A., Meseguer, J.: Security policies and security models. In: Proceedings of S&P, pp. 11–20. IEEE Computer Society (1982). <https://doi.org/10.1109/SP.1982.10014>
21. Hahn, C.: Algorithms for monitoring hyperproperties. In: Finkbeiner, B., Mariani, L. (eds.) RV 2019. LNCS, vol. 11757, pp. 70–90. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32079-9_5
22. Halpern, J.Y., Vardi, M.Y.: The complexity of reasoning about knowledge and time. i. lower bounds. *J. Comput. Syst. Sci.* 38(1), 195–237 (1989). [https://doi.org/10.1016/0022-0000\(89\)90039-1](https://doi.org/10.1016/0022-0000(89)90039-1)
23. Kaivola, R.: Using automata to characterise fixed point temporal logics. Ph.D. thesis (1997)
24. Kupferman, O., Piterman, N., Vardi, M.Y.: From liveness to promptness. *Formal Methods Syst. Des.* 34(2), 83–103 (2009). <https://doi.org/10.1007/s10703-009-0067-z>
25. Nguyen, L.V., Kapinski, J., Jin, X., Deshmukh, J.V., Johnson, T.T.: Hyperproperties of real-valued signals. In: Proceedings of MEMOCODE, pp. 104–113. ACM (2017). <https://doi.org/10.1145/3127041.3127058>
26. Pnueli, A.: The temporal logic of programs. In: Proceedings of FOCS, pp. 46–57. IEEE Computer Society (1977). <https://doi.org/10.1109/SFCS.1977.32>
27. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Proceedings of FOCS, pp. 746–757. IEEE Computer Society (1990). <https://doi.org/10.1109/FSCS.1990.89597>
28. Post, E.L.: A variant of a recursively unsolvable problem. *Bull. Am. Math. Soc.* 52(4), 264–268 (1946)
29. Rabe, M.N.: A temporal logic approach to information-flow control. Ph.D. thesis, Saarland University (2016)
30. Sistla, A.P., Vardi, M.Y., Wolper, P.: The complementation problem for Büchi automata with applications to temporal logic. In: Brauer, W. (ed.) ICALP 1985. LNCS, vol. 194, pp. 465–474. Springer, Heidelberg (1985). <https://doi.org/10.1007/BFb0015772>
31. Sistla, A.P.: Theoretical issues in the design and verification of distributed systems, Ph.D. thesis (1983)
32. Stucki, S., Sánchez, C., Schneider, G., Bonakdarpour, B.: Gray-box monitoring of hyperproperties. In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) FM 2019. LNCS, vol. 11800, pp. 406–424. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30942-8_25
33. Tentrup, L., Weinert, A., Zimmermann, M.: Approximating optimal bounds in prompt-ltl realizability in doubly-exponential time. In: Proceedings of GandALF, EPTCS, vol. 226, pp. 302–315 (2016). <https://doi.org/10.4204/EPTCS.226.21>
34. Zdancewic, S., Myers, A.C.: Observational determinism for concurrent program security. In: Proceedings of CSFW, p. 29. IEEE Computer Society (2003). <https://doi.org/10.1109/CSFW.2003.1212703>

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

