Second-Order Hyperproperties

Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Niklas Metzger

CISPA Helmholtz Center for Information Security,
Saarbrücken, Germany
{raven.beutner,finkbeiner,hadar.frenkel,
niklas.metzger} @cispa.de

Abstract. We introduce Hyper^{2}LTL, a temporal logic for the specification of hyperproperties that allows for second-order quantification over sets of traces. Unlike first-order temporal logics for hyperproperties, such as HyperLTL, Hyper^{2}LTL can express complex epistemic properties like common knowledge, Mazurkiewicz trace theory, and asynchronous hyperproperties. The model checking problem of Hyper^{2}LTL is, in general, undecidable. For the expressive fragment where second-order quantification is restricted to smallest and largest sets, we present an approximate model-checking algorithm that computes increasingly precise under- and overapproximations of the quantified sets, based on fixpoint iteration and automata learning. We report on encouraging experimental results with our model-checking algorithm, which we implemented in the tool `HySO`.

1 Introduction

About a decade ago, Clarkson and Schneider coined the term *hyperproperties* [20] for the rich class of system requirements that relate multiple computations. In their definition, hyperproperties generalize trace properties, which are sets of traces, to *sets of* sets of traces. This covers a wide range of requirements, from information-flow security policies to epistemic properties describing the knowledge of agents in a distributed system. Missing from Clarkson and Schneider's original theory was, however, a concrete specification language that could express customized hyperproperties for specific applications and serve as the common semantic foundation for different verification methods.

A first milestone towards such a language was the introduction of the temporal logic HyperLTL [19]. HyperLTL extends linear-time temporal logic (LTL) with quantification over traces. Suppose, for example, that an agent i in a distributed system observes only a subset of the system variables. The agent *knows* that some LTL formula φ is true on some trace π iff φ holds on *all* traces π' that agent i cannot distinguish from π. If we denote the indistinguishability of π and π' by $\pi \sim_i \pi'$, then the property that *there exists a trace* π *where agent* i *knows* φ can be expressed as the HyperLTL formula

$$\exists \pi. \forall \pi'. \, \pi \sim_i \pi' \rightarrow \varphi(\pi'),$$

where we write $\varphi(\pi')$ to denote that the trace property φ holds on trace π'.

While HyperLTL and its variations have found many applications [30,42,27], the expressiveness of these logics is limited, leaving many widely used hyperproperties out of reach. A prominent example is *common knowledge*, which is used in distributed applications to ensure simultaneous action [29,38]. Common knowledge in a group of agents means that the agents not only know *individually* that some condition φ is true, but that this knowledge is "common" to the group in the sense that each agent *knows* that every agent *knows* that φ is true; on top of that, each agent in the group *knows* that every agent *knows* that every agent *knows* that φ is true; and so on, forming an infinite chain of knowledge.

The fundamental limitation of HyperLTL that makes it impossible to express properties like common knowledge is that the logic is restricted to *first-order quantification*. HyperLTL, then, cannot reason about sets of traces directly, but must always do so by referring to individual traces that are chosen existentially or universally from the full set of traces. For the specification of an agent's individual knowledge, where we are only interested in the (non-)existence of a single trace that is indistinguishable and that violates φ, this is sufficient; however, expressing an infinite chain, as needed for common knowledge, is impossible.

In this paper, we introduce Hyper^{2}LTL, a temporal logic for hyperproperties with *second-order quantification* over traces. In Hyper^{2}LTL, the existence of a trace π where the condition φ is common knowledge can be expressed as the following formula (using slightly simplified syntax):

$$\exists \pi.\, \exists X.\; \pi \in X \wedge \Big(\forall \pi' \in X.\, \forall \pi''.\, \big(\bigvee_{i=1}^{n} \pi' \sim_i \pi''\big) \rightarrow \pi'' \in X\Big) \wedge \forall \pi' \in X.\, \varphi(\pi').$$

The second-order quantifier $\exists X$ postulates the existence of a set X of traces that (1) contains π; that (2) is closed under the observations of each agent, i.e., for every trace π' already in X, all other traces π'' that some agent i cannot distinguish from π' are also in X; and that (3) only contains traces that satisfy φ. The existence of X is a necessary and sufficient condition for φ being common knowledge on π. In the paper, we show that Hyper^{2}LTL is an elegant specification language for many hyperproperties of interest that cannot be expressed in HyperLTL, including, in addition to epistemic properties like common knowledge, also Mazurkiewicz trace theory and asynchronous hyperproperties.

The model checking problem for Hyper^{2}LTL is much more difficult than for HyperLTL. A HyperLTL formula can be checked by translating the LTL subformula into an automaton and then applying a series of automata transformations, such as self-composition to generate multiple traces, projection for existential quantification, and complementation for negation [30,8]. For Hyper^{2}LTL, the model checking problem is, in general, undecidable. We introduce a method that nevertheless obtains sound results by over- and underapproximating the quantified sets of traces. For this purpose, we study Hyper^{2}LTL$_{\text{fp}}$, a fragment of Hyper^{2}LTL, in which we restrict second-order quantification to the smallest or largest set satisfying some property. For example, to check common knowledge, it suffices to consider the *smallest* set X that is closed under the observations of all agents. This smallest set X is defined by the (monotone) fixpoint operation that adds, in each step, all traces that are indistinguishable to some trace already in X.

We develop an approximate model checking algorithm for Hyper^{2}LTL$_{\text{fp}}$ that uses bidirectional inference to deduce lower and upper bounds on second-order variables, interposed with first-order model checking in the style of HyperLTL. Our procedure is parametric in an oracle that provides (increasingly precise) lower and upper bounds. In the paper, we realize the oracles with *fixpoint iteration* for underapproximations of the sets of traces assigned to the second-order variables, and *automata learning* for overapproximations. We report on encouraging experimental results with our model-checking algorithm, which has been implemented in a tool called `HySO`.

2 Preliminaries

For $n \in \mathbb{N}$ we define $[n] := \{1, \ldots, n\}$. We assume that AP is a finite set of atomic propositions and define $\Sigma := 2^{\text{AP}}$. For $t \in \Sigma^\omega$ and $i \in \mathbb{N}$ define $t(i) \in \Sigma$ as the ith element in t (starting with the 0th); and $t[i, \infty]$ for the infinite suffix starting at position i. For traces $t_1, \ldots, t_n \in \Sigma^\omega$ we write $zip(t_1, \ldots, t_n) \in (\Sigma^n)^\omega$ for the pointwise zipping of the traces, i.e., $zip(t_1, \ldots, t_n)(i) := (t_1(i), \ldots, t_n(i))$.

Transition systems. A *transition system* is a tuple $\mathcal{T} = (S, S_0, \kappa, L)$ where S is a set of states, $S_0 \subseteq S$ is a set of initial states, $\kappa \subseteq S \times S$ is a transition relation, and $L : S \to \Sigma$ is a labeling function. A path in $\mathcal{T}$ is an infinite state sequence $s_0 s_1 s_2 \cdots \in S^\omega$, s.t., $s_0 \in S_0$, and $(s_i, s_{i+1}) \in \kappa$ for all i. The associated trace is given by $L(s_0)L(s_1)L(s_2) \cdots \in \Sigma^\omega$ and $\mathit{Traces}(\mathcal{T}) \subseteq \Sigma^\omega$ denotes all traces of $\mathcal{T}$.

Automata. A *non-deterministic Büchi automaton* (NBA) [17] is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ where Σ is a finite alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $F \subseteq Q$ is a set of accepting states, and $\delta : Q \times \Sigma \to 2^Q$ is the transition function. A run on a word $u \in \Sigma^\omega$ is an infinite sequence of states $q_0 q_1 q_2 \cdots \in Q^\omega$ such that $q_0 \in Q_0$ and for every $i \in \mathbb{N}$, $q_{i+1} \in \delta(q_i, u(i))$. The run is accepting if it visits states in F infinitely many times, and we define the language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\omega$, as all infinite words on which $\mathcal{A}$ has an accepting run.

HyperLTL. HyperLTL [19] is one of the most studied temporal logics for the specification of hyperproperties. We assume that $\mathcal{V}$ is a fixed set of trace variables. For the most part, we use variations of π (e.g., $\pi, \pi', \pi_1, \ldots$) to denote trace variables. HyperLTL formulas are then generated by the grammar

$$\begin{aligned}
\varphi &:= \mathbb{Q}\pi.\, \varphi \mid \psi \\
\psi &:= a_\pi \mid \neg\psi \mid \psi \wedge \psi \mid \bigcirc \psi \mid \psi \,\mathcal{U}\, \psi
\end{aligned}$$

where $a \in \text{AP}$ is an atomic proposition, $\pi \in \mathcal{V}$ is a trace variable, $\mathbb{Q} \in \{\forall, \exists\}$ is a quantifier, and $\bigcirc$ and $\mathcal{U}$ are the temporal operators *next* and *until*.

The semantics of HyperLTL is given with respect to a *trace assignment* Π, which is a partial mapping $\Pi : \mathcal{V} \rightharpoonup \Sigma^\omega$ that maps trace variables to traces. Given $\pi \in \mathcal{V}$ and $t \in \Sigma^\omega$ we define $\Pi[\pi \mapsto t]$ as the updated assignment that maps π to t. For $i \in \mathbb{N}$ we define $\Pi[i, \infty]$ as the trace assignment defined by $\Pi[i, \infty](\pi) := \Pi(\pi)[i, \infty]$, i.e., we (synchronously) progress all traces by i steps. For quantifier-free formulas ψ we follow the LTL semantics and define

$$\begin{array}{lll}
\Pi \models a_\pi & \text{iff} & a \in \Pi(\pi)(0) \\
\Pi \models \neg\psi & \text{iff} & \Pi \not\models \psi \\
\Pi \models \psi_1 \land \psi_2 & \text{iff} & \Pi \models \psi_1 \text{ and } \Pi \models \psi_2 \\
\Pi \models \bigcirc \psi & \text{iff} & \Pi[1, \infty] \models \psi \\
\Pi \models \psi_1 \, \mathcal{U} \, \psi_2 & \text{iff} & \exists i \in \mathbb{N}. \, \Pi[i, \infty] \models \psi_2 \text{ and } \forall j < i. \, \Pi[j, \infty] \models \psi_1 \, .
\end{array}$$

The indexed atomic propositions refer to a specific path in Π, i.e., a_π holds iff a holds on the trace bound to π. Quantifiers range over system traces:

$$\Pi \models_{\mathcal{T}} \psi \text{ iff } \Pi \models \psi \qquad \text{and} \qquad \Pi \models_{\mathcal{T}} \mathbb{Q}\pi. \, \varphi \text{ iff } \mathbb{Q}t \in \mathit{Traces}(\mathcal{T}). \, \Pi[\pi \mapsto t] \models \varphi \, .$$

We write $\mathcal{T} \models \varphi$ if $\emptyset \models_{\mathcal{T}} \varphi$ where $\emptyset$ denotes the empty trace assignment.

HyperQPTL. HyperQPTL [43] adds – on top of the trace quantification of HyperLTL – also propositional quantification (analogous to the propositional quantification that QPTL [44] adds on top of LTL). For example, HyperQPTL can express a promptness property which states that there must exist a bound (which is common among all traces), up to which an event must have happened. We can express this as $\exists q. \forall \pi. \diamondsuit q \land (\neg q) \, \mathcal{U} \, a_\pi$ which states that there exists an evaluation of proposition q such that (1) q holds at least once, and (2) for all traces π, a holds on π before the first occurrence of q. See [8] for details.

3 Second-Order HyperLTL

The (first-order) trace quantification in HyperLTL ranges over the set of all system traces; we thus cannot reason about arbitrary sets of traces as required for, e.g., common knowledge. We introduce a second-order extension of HyperLTL by introducing second-order variables (ranging over sets of traces) and allowing quantification over traces from any such set. We present two variants of our logic that differ in the way quantification is resolved. In Hyper^{2}LTL, we quantify over arbitrary sets of traces. While this yields a powerful and intuitive logic, second-order quantification is inherently non-constructive. During model checking, there thus does not exist an efficient way to even approximate possible witnesses for the sets of traces. To solve this quandary, we restrict Hyper^{2}LTL to Hyper^{2}LTL$_{\text{fp}}$, where we instead quantify over sets of traces that satisfy some minimality or maximality constraint. This allows for large fragments of Hyper^{2}LTL$_{\text{fp}}$ that admit algorithmic approximations to its model checking (by, e.g., using known techniques from fixpoint computations [45,46]).

3.1 Hyper^{2}LTL

Alongside the set $\mathcal{V}$ of trace variables, we use a set $\mathfrak{V}$ of second-order variables (which we, for the most part, denote with capital letters $X, Y, ...$). We assume that there is a special variable $\mathfrak{S} \in \mathfrak{V}$ that refers to the set of traces of the given system at hand, and a variable $\mathfrak{A} \in \mathfrak{V}$ that refers to the set of all traces. We define the Hyper^{2}LTL syntax by the following grammar:

$$\begin{aligned}\varphi &:= \mathbb{Q}\pi \in X.\, \varphi \mid \mathbb{Q}X.\, \varphi \mid \psi \\ \psi &:= a_\pi \mid \neg\psi \mid \psi \wedge \psi \mid \bigcirc\psi \mid \psi\,\mathcal{U}\,\psi\end{aligned}$$

where $a \in \mathrm{AP}$ is an atomic proposition, $\pi \in \mathcal{V}$ is a trace variable, $X \in \mathfrak{V}$ is a second-order variable, and $\mathbb{Q} \in \{\forall, \exists\}$ is a quantifier. We also consider the usual derived Boolean constants (*true*, *false*) and connectives ($\vee$, $\rightarrow$, $\leftrightarrow$) as well as the temporal operators *eventually* ($\Diamond\psi := \mathit{true}\,\mathcal{U}\,\psi$) and *globally* ($\Box\psi := \neg\Diamond\neg\psi$). Given a set of atomic propositions $P \subseteq \mathrm{AP}$ and two trace variables π, π', we abbreviate $\pi =_P \pi' := \bigwedge_{a\in P}(a_\pi \leftrightarrow a_{\pi'})$.

Semantics. Apart from a trace assignment Π (as in the semantics of HyperLTL), we maintain a second-order assignment $\Delta : \mathfrak{V} \rightharpoonup 2^{\Sigma^\omega}$ mapping second-order variables to *sets of traces*. Given $X \in \mathfrak{V}$ and $A \subseteq \Sigma^\omega$ we define the updated assignment $\Delta[X \mapsto A]$ as expected. Quantifier-free formulas ψ are then evaluated in a fixed trace assignment as for HyperLTL (cf. Section 2). For the quantifier prefix we define:

$$\begin{aligned}&\Pi, \Delta \models \psi && \text{iff} \quad \Pi \models \psi \\ &\Pi, \Delta \models \mathbb{Q}\pi \in X.\, \varphi && \text{iff} \quad \mathbb{Q}t \in \Delta(X).\, \Pi[\pi \mapsto t], \Delta \models \varphi \\ &\Pi, \Delta \models \mathbb{Q}X.\, \varphi && \text{iff} \quad \mathbb{Q}A \subseteq \Sigma^\omega.\, \Pi, \Delta[X \mapsto A] \models \varphi\end{aligned}$$

Second-order quantification updates Δ with a set of traces, and first-order quantification updates Π by quantifying over traces within the set defined by Δ.

Initially, we evaluate a formula in the empty trace assignment and fix the valuation of the special second-order variable $\mathfrak{S}$ to be the set of all system traces and $\mathfrak{A}$ to be the set of all traces. That is, given a system $\mathcal{T}$ and Hyper^{2}LTL formula φ, we say that $\mathcal{T}$ satisfies φ, written $\mathcal{T} \models \varphi$, if $\emptyset, [\mathfrak{S} \mapsto \mathit{Traces}(\mathcal{T}), \mathfrak{A} \mapsto \Sigma^\omega] \models \varphi$, where we write $\emptyset$ for the empty trace assignment. The model-checking problem for Hyper^{2}LTL is checking whether $\mathcal{T} \models \varphi$ holds.

Hyper^{2}LTL naturally generalizes HyperLTL by adding second-order quantification. As sets range over *arbitrary* traces, Hyper^{2}LTL also subsumes the more powerful logic HyperQPTL. The proof of Lemma 1 is given in Appendix A.1.

Lemma 1. Hyper^{2}LTL *subsumes* HyperQPTL *(and thus also* HyperLTL*).*

Syntactic Sugar. In Hyper^{2}LTL, we can quantify over traces within a second-order variable, but we cannot state, within the body of the formula, that some path is a member of some second-order variable. For that, we define $\pi \triangleright X$ (as an atom within the body) as syntactic sugar for $\exists \pi' \in X. \Box(\pi' =_{\mathrm{AP}} \pi)$, i.e., π is in X if there exists some trace in X that agrees with π on all propositions. Note that we can only use $\pi \triangleright X$ *outside* of the scope of any temporal operators; this ensures that we can bring the resulting formula into a form that conforms to the Hyper^{2}LTL syntax.

3.2 Hyper^{2}LTL$_{fp}$

The semantics of Hyper^{2}LTL quantifies over arbitrary sets of traces, making even approximations to its semantics challenging. We propose Hyper^{2}LTL$_{\mathrm{fp}}$ as a restriction that only quantifies over sets that are subject to an additional minimality or maximality constraint. For large classes of formulas, we show that this admits effective model-checking approximations. We define Hyper^{2}LTL$_{\mathrm{fp}}$ by the following grammar:

$$\varphi := \mathbb{Q}\, \pi \in X. \, \varphi \mid \mathbb{Q}\, (X, \Join, \varphi). \, \varphi \mid \psi$$
$$\psi := a_\pi \mid \neg \psi \mid \psi \wedge \psi \mid \bigcirc \psi \mid \psi \, \mathcal{U} \, \psi$$

where $a \in \mathrm{AP}$, $\pi \in \mathcal{V}$, $X \in \mathfrak{V}$, $\mathbb{Q} \in \{\forall, \exists\}$, and $\Join \in \{\curlywedge, \curlyvee\}$ determines if we consider smallest ($\curlyvee$) or largest ($\curlywedge$) sets. For example, the formula $\exists\, (X, \curlyvee, \varphi_1). \, \varphi_2$ holds if there exists some set of traces X, that satisfies both φ_1 and φ_2, and is *a* smallest set that satisfies φ_1. Such minimality and maximality constraints with respect to a (hyper)property arise naturally in many properties. Examples include common knowledge (cf. Section 3.3), asynchronous hyperproperties (cf. Section 4.2), and causality in reactive systems [22,21].

Semantics. For path formulas, the semantics of Hyper^{2}LTL$_{\mathrm{fp}}$ is defined analogously to that of Hyper^{2}LTL and HyperLTL. For the quantifier prefix we define:

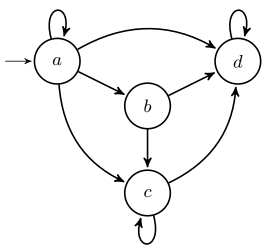
$$\begin{array}{lll} \Pi, \Delta \models \psi & \text{iff} & \Pi \models \psi \\ \Pi, \Delta \models \mathbb{Q}\pi \in X. \, \varphi & \text{iff} & \mathbb{Q} t \in \Delta(X). \, \Pi[\pi \mapsto t], \Delta \models \varphi \\ \Pi, \Delta \models \mathbb{Q}(X, \Join, \varphi_1). \, \varphi_2 & \text{iff} & \mathbb{Q} A \in \mathit{sol}(\Pi, \Delta, (X, \Join, \varphi_1)). \, \Pi, \Delta[X \mapsto A] \models \varphi_2 \end{array}$$

where $\mathit{sol}(\Pi, \Delta, (X, \Join, \varphi_1))$ denotes all solutions to the minimality/maximality condition given by φ_1, which we define by mutual recursion as follows:

$$\mathit{sol}(\Pi, \Delta, (X, \curlyvee, \varphi)) := \{A \subseteq \Sigma^\omega \mid \Pi, \Delta[X \mapsto A] \models \varphi \wedge \forall A' \subsetneq A. \, \Pi, \Delta[X \mapsto A'] \not\models \varphi\}$$
$$\mathit{sol}(\Pi, \Delta, (X, \curlywedge, \varphi)) := \{A \subseteq \Sigma^\omega \mid \Pi, \Delta[X \mapsto A] \models \varphi \wedge \forall A' \supsetneq A. \, \Pi, \Delta[X \mapsto A'] \not\models \varphi\}$$

A set A satisfies the minimality/maximality constraint if it satisfies φ and is a least (in case $\Join = \curlyvee$) or greatest (in case $\Join = \curlywedge$) set that satisfies φ.

Note that $\mathit{sol}(\Pi, \Delta, (X, \Join, \varphi))$ can contain multiple sets or no set at all, i.e., there may not exists a unique least or greatest set that satisfies φ. In Hyper^{2}LTL$_{\mathrm{fp}}$, we therefore add an additional quantification over the set of all



$$\begin{aligned}
\pi &= a^n d^\omega \\
K_2(\pi) &= a^{n-1} b d^\omega \\
K_1 K_2(\pi) &= a^{n-1} c d^\omega \\
K_2 K_1 K_2(\pi) &= a^{n-2} b c d^\omega \\
&\dots \\
K_1 K_2 \dots K_2(\pi) &= a c^{n-1} d^\omega
\end{aligned}$$

Fig. 1: Left: An example for a multi-agent system with two agents, where agent 1 observes a and d, and agent 2 observes c and d. Right: The iterative construction of the traces to be considered for common knowledge starting with $a^n d^\omega$.

solutions to the minimality/maximality constraint. When discussing our model checking approximation algorithm, we present a (syntactic) restriction on φ which guarantees that $sol(\Pi, \Delta, (X, \mathfrak{X}, \varphi))$ contains a unique element (i.e., is a singleton set). Moreover, our restriction allows us to employ fixpoint techniques to find approximations to this unique solution. In case the solution for $(X, \mathfrak{X}, \varphi)$ is unique, we often omit the leading quantifier and simply write $(X, \mathfrak{X}, \varphi)$ instead of $\mathbb{Q}(X, \mathfrak{X}, \varphi)$.

As we can encode the minimality/maximality constraints of Hyper^{2}LTL$_{\text{fp}}$ in Hyper^{2}LTL (see Appendix A.2), we have the following:

Proposition 1. *Any* Hyper^{2}LTL$_{\text{fp}}$ *formula φ can be effectively translated into an* Hyper^{2}LTL *formula φ' such that for all transition systems $\mathcal{T}$ we have $\mathcal{T} \models \varphi$ iff $\mathcal{T} \models \varphi'$.*

3.3 Common Knowledge in Multi-Agent Systems

To explain common knowledge, we use a variation of an example from [41], and encode it in Hyper^{2}LTL$_{\text{fp}}$. Figure 1(left) shows a transition system of a distributed system with two agents, agent 1 and agent 2. Agent 1 observes variables a and d, whereas agent 2 observes c and d. The property of interest is *starting from the trace $\pi = a^n d^\omega$ for some fixed $n > 1$, is it common knowledge for the two agents that a holds in the second step.* It is trivial to see that $\bigcirc a$ holds on π. However, for common knowledge, we consider the (possibly) infinite chain of observationally equivalent traces. For example, agent 2 cannot distinguish the traces $a^n d^\omega$ and $a^{n-1} b d^\omega$. Therefore, agent 2 only knows that $\bigcirc a$ holds on π if it also holds on $\pi' = a^{n-1} b d^\omega$. For common knowledge, agent 1 also has to know that agent 2 knows $\bigcirc a$, which means that for all traces that are indistinguishable from π or π' for agent 1, $\bigcirc a$ has to hold. This adds $\pi'' = a^{n-1} c d^\omega$ to the set of traces to verify $\bigcirc a$ against. This chain of reasoning continues as shown in Figure 1(right). In the last step we add $a c^{n-1} d^\omega$ to the set of indistinguishable traces, concluding that $\bigcirc a$ is not common knowledge.

The following Hyper^{2}LTL$_{\text{fp}}$ formula specifies the property stated above. The abbreviation $obs(\pi_1, \pi_2) := \Box(\pi_1 =_{\{a,d\}} \pi_2) \vee \Box(\pi_1 =_{\{c,d\}} \pi_2)$ denotes that π_1 and π_2 are observationally equivalent for either agent 1 or agent 2.

$$\forall \pi \in \mathfrak{S}.\big(\bigwedge_{i=0}^{n-1} \bigcirc^i a_\pi \wedge \bigcirc^n \Box d_\pi \big) \rightarrow$$
$$\Big(X, \curlyvee, \pi \triangleright X \wedge \big(\forall \pi_1 \in X. \forall \pi_2 \in \mathfrak{S}.\, obs(\pi_1, \pi_2) \rightarrow \pi_2 \triangleright X\big)\Big). \forall \pi' \in X. \bigcirc a_{\pi'}$$

For a trace π of the form $\pi = a^n d^\omega$, the set X represents the *common knowledge set* on π. This set X is the smallest set that (1) contains π (expressed using our syntactic sugar $\triangleright$); and (2) is closed under observations by either agent, i.e., if we find some $\pi_1 \in X$ and some system trace π_2 that are observationally equivalent, π_2 should also be in X. Note that this set is unique (due to the minimality restriction), so we do not quantify it explicitly. Lastly, we require that all traces in X satisfy the property $\bigcirc a$. All sets that satisfy this formula would also include the trace $ac^{n-1}d^\omega$, and therefore no such X exists; thus, we can conclude that starting from trace $a^n d^\omega$, it is *not* common knowledge that $\bigcirc a$ holds.

On the other hand, it *is* common knowledge that a holds in the *first* step (cf. Section 6).

3.4 Hyper^{2}LTL Model Checking

As Hyper^{2}LTL and Hyper^{2}LTL$_{\text{fp}}$ allow quantification over arbitrary sets of traces, we can encode the satisfiability of HyperQPTL (i.e., the question of whether some set of traces satisfies a formula) within their model-checking problem; rendering the model-checking problem highly undecidable [32], even for very simple formulas [4].

Proposition 2. *For any* HyperQPTL *formula φ there exists a* Hyper^{2}LTL *formula φ' such that φ is satisfiable iff φ' holds on some arbitrary transition system. The model-checking problem of* Hyper^{2}LTL *is thus highly undecidable (Σ_1^1-hard).*

Proof. Let φ' be the Hyper^{2}LTL formula obtained from φ by replacing each HyperQPTL trace quantifier $\mathbb{Q}\pi$ with the Hyper^{2}LTL quantifier $\mathbb{Q}\pi \in X$, and each propositional quantifier $\mathbb{Q}q$ with $\mathbb{Q}\pi_q \in \mathfrak{A}$ for some fresh trace variable π_q. In the body, we replace each propositional variable q with a_{π_q} for some fixed proposition $a \in \text{AP}$. Then, φ is satisfiable iff the Hyper^{2}LTL formula $\exists X. \varphi'$ holds in some arbitrary system. $\square$

Hyper^{2}LTL$_{\text{fp}}$ cannot express HyperQPTL satisfiability directly. If there exists a model of a HyperQPTL formula, there may not exist a least one. However, model checking of Hyper^{2}LTL$_{\text{fp}}$ is also highly undecidable.

Proposition 3. *The model-checking problem of* Hyper^{2}LTL$_{\text{fp}}$ *is Σ_1^1-hard.*

Proof (Sketch). We can encode the existence of a *recurrent* computation of a Turing machine, which is known to be Σ_1^1-hard [1]. □

Conversely, the *existential* fragment of Hyper^{2}LTL can be encoded back into HyperQPTL satisfiability:

Proposition 4. *Let φ be a* Hyper^{2}LTL *formula that uses only existential second-order quantification and $\mathcal{T}$ be any system. We can effectively construct a formula φ' in* HyperQPTL *such that $\mathcal{T} \models \varphi$ iff φ' is satisfiable.*

Lastly, we present some easy fragments of Hyper^{2}LTL for which the model-checking problem is decidable. Here we write $\exists^* X$ (resp. $\forall^* X$) for some sequence of existentially (resp. universally) quantified *second-order* variables and $\exists^*\pi$ (resp. $\forall^*\pi$) for some sequence of existentially (resp. universally) quantified *first-order* variables. For example, $\exists^* X \forall^* \pi$ captures all formulas of the form $\exists X_1, \dots X_n.\forall \pi_1, \dots, \pi_m.\psi$ where ψ is quantifier-free.

Proposition 5. *The model-checking problem of* Hyper^{2}LTL *is decidable for the fragments: $\exists^* X \forall^* \pi$, $\forall^* X \forall^* \pi$, $\exists^* X \exists^* \pi$, $\forall^* X \exists^* \pi$, $\exists X.\exists^* \pi \in X \forall^* \pi' \in X$.*

See Appendix A.3 for the full proofs of the propositions above.

4 Expressiveness of Hyper^{2}LTL

In this section, we point to existing logics that can naturally be encoded within our second-order hyperlogics Hyper^{2}LTL and Hyper^{2}LTL$_{\text{fp}}$.

4.1 Hyper^{2}LTL and LTL$_{\mathsf{K,C}}$

LTL$_{\mathsf{K}}$ extends LTL with the knowledge operator K. For some subset of agents A, the formula $\mathsf{K}_A\psi$ holds in timestep i, if ψ holds on all traces equivalent to some agent in A up to timestep i. See Appendix B.1 for detailed semantics. LTL$_{\mathsf{K}}$ and HyperCTL* have incomparable expressiveness [15] but the knowledge operator K can be encoded by either adding a linear past operator [15] or by adding propositional quantification (as in HyperQPTL) [43].

Using Hyper^{2}LTL$_{\text{fp}}$ we can encode LTL$_{\mathsf{K,C}}$, featuring the knowledge operator K *and* the common knowledge operator C (which requires that ψ holds on the closure set of equivalent traces, up to the current timepoint) [39]. Note that LTL$_{\mathsf{K,C}}$ is not encodable by only adding propositional quantification or the linear past operator.

Proposition 6. *For every* LTL$_{\mathsf{K,C}}$ *formula φ there exists an* Hyper^{2}LTL$_{\text{fp}}$ *formula φ' such that for any system $\mathcal{T}$ we have $\mathcal{T} \models_{LTL_{\mathsf{K,C}}} \varphi$ iff $\mathcal{T} \models \varphi'$.*

Proof (Sketch). We follow the intuition discussed in Section 3.3. For each occurrence of a knowledge operator in $\{\mathsf{K}, \mathsf{C}\}$, we introduce a second-order set that collects all equivalent traces, and a fresh trace variable that keeps track on the points in time with respect to which we need to compare. We then inductively construct a Hyper^{2}LTL$_{\text{fp}}$ formula that captures all the knowledge and common-knowledge sets. For more details see Appendix B.1. □

4.2 Hyper^{2}LTL and Asynchronous Hyperproperties

Most existing hyperlogics (including Hyper^{2}LTL) traverse the traces of a system *synchronously.* However, in many cases such a synchronous traversal is too restricting and we need to compare traces asynchronously. As an example, consider *observational determinism* (OD), which we can express in HyperLTL as $\varphi_{OD} := \forall \pi_1. \forall \pi_2. \Box(o_{\pi_1} \leftrightarrow o_{\pi_2})$. The formula states that the output of a system is identical across all traces and so (trivially) no information about high-security inputs is leaked. In most systems encountered in practice, this synchronous formula is violated, as the exact timing between updates to o might differ by a few steps (see Appendix B.2 for some examples). However, assuming that an attacker only has access to the memory footprint and not a timing channel, we would only like to check that all traces are *stutter* equivalent (with respect to o).

A range of extensions to existing hyperlogics has been proposed to reason about such asynchronous hyperproperties [3,16,37,5,9]. We consider AHLTL [3]. An AHLTL formula has the form $\mathbb{Q}_1 \pi_1, \dots, \mathbb{Q}_n \pi_m. \mathbf{E}. \psi$ where ψ is a qunatifier-free HyperLTL formula. The initial trace quantifier prefix is handled as in HyperLTL. However, different from HyperLTL, a trace assignment $[\pi_1 \mapsto t_1, \dots, \pi_n \mapsto t_n]$ satisfies $\mathbf{E}. \psi$ if there exist stuttered traces $t'_1, \dots, t'_n$ of $t_1, \dots, t_n$ such that $[\pi_1 \mapsto t'_1, \dots, \pi_n \mapsto t'_n] \models \psi$. We write $\mathcal{T} \models_{AHLTL} \varphi$ if a system $\mathcal{T}$ satisfies the AHLTL formula φ. Using this quantification over stutterings we can, for example, express an asynchronous version of observational determinism as $\forall \pi_1. \forall \pi_2. \mathbf{E}. \Box(o_{\pi_1} \leftrightarrow o_{\pi_2})$ stating that every two traces can be aligned such that they (globally) agree on o. Despite the fact that Hyper^{2}LTL$_{\text{fp}}$ is itself synchronous, we can use second-order quantification to encode asynchronous hyperproperties, as we state in the following proposition.

Proposition 7. *For any AHLTL formula φ there exists a* Hyper^{2}LTL$_{\text{fp}}$ *formula φ' such that for any system $\mathcal{T}$ we have $\mathcal{T} \models_{AHLTL} \varphi$ iff $\mathcal{T} \models \varphi'$.*

Proof. Assume that $\varphi = \mathbb{Q}_1 \pi_1, \dots, \mathbb{Q}_n \pi_n. \mathbf{E}. \psi$ is the given AHLTL formula. For each $i \in [n]$ we define a formula φ_i as follows

$$\forall \pi_1 \in X_i. \forall \pi_2 \in \mathfrak{A}. \Big((\pi_1 =_{\text{AP}} \pi_2) \, \mathcal{U} \, (\Box \bigwedge_{a \in \text{AP}} a_{\pi_1} \leftrightarrow \bigcirc a_{\pi_2}) \Big) \rightarrow \pi_2 \triangleright X_i$$

The formula asserts that the set of traces bound to X_i is closed under stuttering, i.e., if we start from any trace in X_i and stutter it once (at some arbitrary position) we again end up in X_i. Using the formulas φ_i, we then construct a Hyper^{2}LTL$_{\text{fp}}$ formula that is equivalent to φ as follows

$$\begin{aligned} \varphi' := \mathbb{Q}_1 \pi_1 \in \mathfrak{S}, \dots, \mathbb{Q}_n \pi_n \in \mathfrak{S}. (X_1, \curlyvee, \pi_1 \triangleright X_1 \land \varphi_1) \cdots (X_n, \curlyvee, \pi_n \triangleright X_n \land \varphi_n) \\ \exists \pi'_1 \in X_1, \dots, \exists \pi'_n \in X_n. \psi[\pi'_1/\pi_1, \dots, \pi'_n/\pi_n] \end{aligned}$$

We first mimic the quantification in φ and, for each trace π_i, construct a least set X_i that contains π_i and is closed under stuttering (thus describing exactly the set of all stuttering of π_i). Finally, we assert that there are traces $\pi'_1, \dots, \pi'_n$ with $\pi'_i \in X_i$ (so π'_i is a stuttering of π_i) such that $\pi'_1, \dots, \pi'_n$ satisfy ψ. It is easy to see that $\mathcal{T} \models_{AHLTL} \varphi$ iff $\mathcal{T} \models \varphi'$ holds for all systems. $\Box$

Hyper^{2}LTL$_{\text{fp}}$ captures all properties expressible in AHLTL. In particular, our approximate model-checking algorithm for Hyper^{2}LTL$_{\text{fp}}$ (cf. Section 5) is applicable to AHLTL; even for instances where no approximate solutions were previously known. In Section 6, we show that our prototype model checker for Hyper^{2}LTL$_{\text{fp}}$ can verify asynchronous properties in practice.

5 Model-Checking Hyper^{2}LTL$_{fp}$

In general, finite-state model checking of Hyper^{2}LTL$_{\text{fp}}$ is highly undecidable (cf. Proposition 2). In this section, we outline a partial algorithm that computes approximations on the concrete values of second-order variables for a fragment of Hyper^{2}LTL$_{\text{fp}}$. At a very high-level, our algorithm (Algorithm 1) iteratively computes under- and overapproximations for second-order variables. It then turns to resolve first-order quantification, using techniques from HyperLTL model checking [30,8], and resolves existential and universal trace quantification on the under- and overapproximation of the second-order variables, respectively. If the verification fails, it goes back to refine second-order approximations.

In this section, we focus on the setting where we are interested in the least sets (using $\curlyvee$), and use techniques to approximate the *least* fixpoint. A similar (dual) treatment is possible for Hyper^{2}LTL$_{\text{fp}}$ formulas that use the largest set. Every Hyper^{2}LTL$_{\text{fp}}$ which uses only minimal sets has the following form:

$$\varphi = \gamma_1.(Y_1, \curlyvee, \varphi_1^{con}).\gamma_2 \ldots .(Y_k, \curlyvee, \varphi_k^{con}).\, \gamma_{k+1}.\, \psi \tag{1}$$

We quantify second-order variables $Y_1, \ldots, Y_k$, where, for each $j \in [k]$, Y_j is the least set that satisfies φ_j^{con}. Finally, for each $j \in [k+1]$,

$$\gamma_j = \mathbb{Q}_{l_j+1}\pi_{l_j+1} \in X_{l_j+1} \ldots \mathbb{Q}_{l_{j+1}}\pi_{l_{j+1}} \in X_{l_{j+1}}$$

is the block of first-order quantifiers that sits between the quantification of Y_{j-1} and Y_j. Here $X_{l_j+1}, \ldots, X_{l_{j+1}} \in \{\mathfrak{S}, \mathfrak{A}, Y_1, \ldots, Y_{j-1}\}$ are second-order variables that are quantified before γ_j. In particular, $\pi_1, \ldots, \pi_{l_j}$ are the first-order variables quantified before Y_j.

5.1 Fixpoints in Hyper^{2}LTL$_{fp}$

We consider a fragment of Hyper^{2}LTL$_{\text{fp}}$ which we call the *least fixpoint fragment*. Within this fragment, we restrict the formulas $\varphi_1^{con}, \ldots, \varphi_k^{con}$ such that $Y_1, \ldots, Y_k$ can be approximated as (least) fixpoints. Concretely, we say that φ is in the *least fixpoint fragment* of Hyper^{2}LTL$_{\text{fp}}$ if for all $j \in [k]$, φ_j^{con} is a conjunction of formulas of the form

$$\forall \dot\pi_1 \in X_1 \ldots \forall \dot\pi_n \in X_n.\, \psi_{step} \rightarrow \dot\pi_M \triangleright Y_j \tag{2}$$

where each $X_i \in \{\mathfrak{S}, \mathfrak{A}, Y_1, \ldots, Y_j\}$, ψ_{step} is quantifier-free formula over trace variables $\dot\pi_1, \ldots, \dot\pi_n, \pi_1, \ldots, \pi_{l_j}$, and $M \in [n]$. Intuitively, Equation (2) states a requirement on traces that should be included in Y_j. If we find traces $\dot{t}_1 \in X_1, \ldots, \dot{t}_n \in X_n$ that, together with the traces $t_1, \ldots, t_{l_j}$ quantified before Y_j, satisfy ψ_{step}, then $\dot{t}_M$ should be included in Y_j.

Together with the minimality constraint on Y_j (stemming from the semantics of Hyper^{2}LTL$_{\text{fp}}$), this effectively defines a (monotone) least fixpoint computation, as ψ_{step} defines exactly the traces to be added to the set. This will allow us to use results from fixpoint theory to compute approximations for the sets Y_j.

Our least fixpoint fragment captures most properties of interest, in particular, common knowledge (Section 3.3) and asynchronous hyperproperties (Section 4.2). We observe that formulas of the above form ensure that the solution Y_j is unique, i.e., for any trace assignment Π to $\pi_1, \ldots, \pi_{l_j}$ and second-order assignment Δ to $\mathfrak{S}, \mathfrak{A}, Y_1, \ldots, Y_{j-1}$, there is only one element in $sol(\Pi, \Delta, (Y_j, \curlyvee, \varphi_j^{con}))$.

5.2 Functions as Automata

In our (approximate) model-checking algorithm, we represent a concrete assignment to the second-order variables $Y_1, \ldots, Y_k$ using automata $\mathcal{B}_{Y_1}, \ldots, \mathcal{B}_{Y_k}$. The concrete assignment of Y_j can depend on traces assigned to $\pi_1, \ldots, \pi_{l_j}$, i.e., the first-order variables quantified before Y_j. To capture these dependencies, we view each Y_j not as a set of traces but as a function mapping traces of all preceding first-order variables to a set of traces. We represent such a function $f : (\Sigma^\omega)^{l_j} \to 2^{(\Sigma^\omega)}$ mapping the l_j traces to a set of traces as an automaton $\mathcal{A}$ over Σ^{l_j+1}. For traces $t_1, \ldots, t_{l_j}$, the set $f(t_1, \ldots, t_{l_j})$ is represented in the automaton by the set $\{t \in \Sigma^\omega \mid zip(t_1, \ldots, t_{l_j}, t) \in \mathcal{L}(\mathcal{A})\}$. For example, the function $f(t_1) := \{t_1\}$ can be defined by the automaton that accepts the zipping of a pair of traces exactly if both traces agree on all propositions. This representation of functions as automata allows us to maintain an assignment to Y_j that is parametric in $\pi_1, \ldots, \pi_{l_j}$ and still allows first-order model checking on $Y_1, \ldots, Y_k$.

5.3 Model Checking for First-Order Quantification

First, we focus on first-order quantification, and assume that we are given a concrete assignment for each second-order variable as fixed automata $\mathcal{B}_{Y_1}, \ldots, \mathcal{B}_{Y_k}$ (where $\mathcal{B}_{Y_j}$ is an automaton over Σ^{l_j+1}). Our construction for resolving first-order quantification is based on HyperLTL model checking [30], but needs to work on sets of traces that, themselves, are based on traces quantified before (cf. Section 5.2). Recall that the first-order quantifier prefix is $\gamma_1 \cdots \gamma_{k+1} = \mathbb{Q}_1 \pi_1 \in X_1 \cdots \mathbb{Q}_{l_{k+1}} \pi_{l_{k+1}} \in X_{l_{k+1}}$. For each $1 \leq i \leq l_{k+1}$ we inductively construct an automaton $\mathcal{A}_i$ over Σ^{i-1} that summarizes all trace assignments to $\pi_1, \ldots, \pi_{i-1}$ that satisfy the subformula starting with the quantification of π_i. That is, for all traces $t_1, \ldots, t_{i-1}$ we have

$$[\pi_1 \mapsto t_1, \ldots, \pi_{i-1} \mapsto t_{i-1}] \vDash \mathbb{Q}_i \pi_i \in X_i \cdots \mathbb{Q}_{l_{k+1}} \pi_{l_{k+1}} \in X_{l_{k+1}}. \psi$$

(under the fixed second-order assignment for $Y_1, \dots, Y_k$ given by $\mathcal{B}_{Y_1}, \dots, \mathcal{B}_{Y_k}$) if and only if $\mathit{zip}(t_1, \dots, t_{i-1}) \in \mathcal{L}(\mathcal{A}_i)$. In the context of HyperLTL model checking we say $\mathcal{A}_i$ is *equivalent* to $\mathbb{Q}_i \pi_i \in X_i \cdots \mathbb{Q}_{l_{k+1}} \pi_{l_{k+1}} \in X_{l_{k+1}}. \psi$ [30,8]. In particular, $\mathcal{A}_1$ is an automaton over singleton alphabet Σ^0.

We construct $\mathcal{A}_1, \dots, \mathcal{A}_{l_{k+1}+1}$ inductively, starting with $\mathcal{A}_{l_{k+1}+1}$. Initially, we construct $\mathcal{A}_{l_{k+1}+1}$ (over $\Sigma^{l_{k+1}}$) using a standard LTL-to-NBA construction on the (quantifier-free) body ψ (see [30] for details). Now assume that we are given an (inductively constructed) automaton $\mathcal{A}_{i+1}$ over Σ^i and want to construct $\mathcal{A}_i$. We first consider the case where $\mathbb{Q}_i = \exists$, i.e., the ith trace quantification is existential. Now X_i (the set where π_i is resolved on) either equals $\mathfrak{S}$, $\mathfrak{A}$ or Y_j for some $j \in [k]$. In either case, we represent the current assignment to X_i as an automaton $\mathcal{C}$ over Σ^{T+1} for some $T < i$ that defines the model of X_i based on traces $\pi_1, \dots, \pi_T$: In case $X_i = \mathfrak{S}$, we set $\mathcal{C}$ to be the automaton over Σ^{0+1} that accepts exactly the traces in the given system $\mathcal{T}$; in case $X_i = \mathfrak{A}$, we set $\mathcal{C}$ to be the automaton over Σ^{0+1} that accepts all traces; If $X_i = Y_j$ for some $j \in [k]$ we set $\mathcal{C}$ to be $\mathcal{B}_{Y_j}$ (which is an automaton over Σ^{l_j+1}).[1] Given $\mathcal{C}$, we can now modify the construction from [30], to resolve first-order quantification: The desired automaton $\mathcal{A}_i$ should accept the zipping of traces $t_1, \dots, t_{i-1}$ if there exists a trace t such that (1) $\mathit{zip}(t_1, \dots, t_{i-1}, t) \in \mathcal{L}(\mathcal{A}_{i+1})$, *and* (2) the trace t is contained in the set of traces assigned to X_i as given by $\mathcal{C}$, i.e., $\mathit{zip}(t_1, \dots, t_T, t) \in \mathcal{L}(\mathcal{C})$. The construction of this automaton is straightforward by taking a product of $\mathcal{A}_{i+1}$ and $\mathcal{C}$. We denote this automaton with `eProduct(`$\mathcal{A}_{i+1}$`,`$\mathcal{C}$`)`. In case $\mathbb{Q}_i = \forall$ we exploit the duality that $\forall \pi. \psi = \neg \exists \pi. \neg \psi$, combining the above construction with automata complementation. We denote this universal product of $\mathcal{A}_{i+1}$ and $\mathcal{C}$ with `uProduct(`$\mathcal{A}_{i+1}$`,`$\mathcal{C}$`)`.

The final automaton $\mathcal{A}_1$ is an automaton over singleton alphabet Σ^0 that is equivalent to $\gamma_1 \cdots \gamma_{k+1}. \psi$, i.e., the entire first-order quantifier prefix. Automaton $\mathcal{A}_1$ thus satisfies $\mathcal{L}(\mathcal{A}_1) \neq \emptyset$ (which we can decide) iff the empty trace assignment satisfies the first-order formula $\gamma_1 \cdots \gamma_{k+1}. \psi$, iff φ (of Equation (1)) holds within the fixed model for $Y_1, \dots, Y_k$. For a given fixed second-order assignment (given as automata $\mathcal{B}_{Y_1}, \dots, \mathcal{B}_{Y_k}$), we can thus decide if the system satisfies the first-order part.

During the first-order model-checking phase, each quantifier alternations in the formula require complex automata complementation. For the first-order phase, we could also use cheaper approximate methods by, e.g., instantiating the existential trace using a strategy [24,7,6].

5.4 Bidirectional Model Checking

So far, we have discussed the verification of the first-order quantifiers assuming we have a fixed model for all second-order variables $Y_1, \dots, Y_k$. In our actual model-checking algorithm, we instead maintain under- and overapproximations on each of the $Y_1, \dots, Y_k$.

[1] Note that in this case $l_j < i$: if trace π_i is resolved on Y_j (i.e, $X_i = Y_j$), then Y_j must be quantified *before* π_i so there are at most $i - 1$ traces quantified before Y_j.

Algorithm 1

```
verify($\varphi$, $\mathcal{T}$) =
  let $\varphi$ = $\left[\gamma_j \; (Y_j, \curlyvee, \varphi_j^{con})\right]_{j=1}^{k} \gamma_{k+1}. \psi$ where $\gamma_i$ = $\left[\mathbb{Q}_m \pi_m \in X_m\right]_{m=l_i+1}^{l_{i+1}}$
  let N = 0
  let $\mathcal{A}_\mathcal{T}$ = systemToNBA($\mathcal{T}$)
  repeat
    // Start outside-in traversal on second-order variables
    let $\flat$ = $\left[\mathfrak{S} \mapsto (\mathcal{A}_\mathcal{T}, \mathcal{A}_\mathcal{T}), \mathfrak{A} \mapsto (\mathcal{A}_\top, \mathcal{A}_\top)\right]$
    for $j$ from 1 to $k$ do
      $\mathcal{B}_j^l$ := underApprox(($Y_j, \curlyvee, \varphi_j^{con}$),$\flat$,$N$)
      $\mathcal{B}_j^u$ := overApprox(($Y_j, \curlyvee, \varphi_j^{con}$),$\flat$,$N$)
      $\flat(Y_j)$ := ($\mathcal{B}_j^l, \mathcal{B}_j^u$)
    // Start inside-out traversal on first-order variables
    let $\mathcal{A}_{l_{k+1}+1}$ = LTLtoNBA($\psi$)
    for $i$ from $l_{k+1}$ to 1 do
      let ($\mathcal{C}^l$,$\mathcal{C}^u$) = $\flat(X_i)$
      if $\mathbb{Q}_i = \exists$ then
        $\mathcal{A}_i$ := eProduct($\mathcal{A}_{i+1}$, $\mathcal{C}^l$)
      else
        $\mathcal{A}_i$ := uProduct($\mathcal{A}_{i+1}$, $\mathcal{C}^u$)
    if $\mathcal{L}(\mathcal{A}_1) \neq \emptyset$ then
      return SAT
   else
      $N$ = $N$ + 1
```

In each iteration, we first traverse the second-order quantifiers in an *outside-in* direction and compute lower- and upper-bounds on each Y_j. Given the bounds, we then traverse the first-order prefix in an *inside-out* direction using the current approximations to $Y_1, \ldots, Y_k$. If the current approximations are not precise enough to witness the satisfaction (or violation) of a property, we repeat and try to compute better bounds on $Y_1, \ldots, Y_k$. Due to the different directions of traversal, we refer to our model-checking approach as *bidirectional*. Algorithm 1 provides an overview. Initially, we convert the system $\mathcal{T}$ to an NBA $\mathcal{A}_\mathcal{T}$ accepting exactly the traces of the system. In each round, we compute under- and overapproximations for each Y_j in a mapping $\flat$. We initialize $\flat$ by mapping $\mathfrak{S}$ to $(\mathcal{A}_\mathcal{T}, \mathcal{A}_\mathcal{T})$ (i.e., the value assigned to the system variable is precisely $\mathcal{A}_\mathcal{T}$ for both under- and overapproximation), and $\mathfrak{A}$ to $(\mathcal{A}_\top, \mathcal{A}_\top)$ where $\mathcal{A}_\top$ is an automaton over Σ^1 accepting all traces. We then traverse the second-order quantifiers outside-in (from Y_1 to Y_k) and for each Y_j compute a pair $(\mathcal{B}_j^l, \mathcal{B}_j^u)$ of automata over Σ^{l_j+1} that under- and overapproximate the actual (unique) model of Y_j. We compute these approximations using functions `underApprox` and `overApprox`, which can be instantiated with any procedure that computes sound lower and upper bounds (see Section 5.5). During verification, we further maintain a precision bound N (initially set to 0) that tracks the current precision of the second-order approximations.

When $\flat$ contains an under- and overapproximation for each second-order variable, we traverse the first-order variables in an inside-out direction (from $\pi_{l_{k+1}}$ to π_1) and, following the construction outlined in Section 5.3, construct automata $\mathcal{A}_{l_k+1}, \ldots, \mathcal{A}_1$. Different from the simplified setting in Section 5.3 (where we assume a fixed automaton $\mathcal{B}_{Y_j}$ providing a model for each Y_j), the mapping $\flat$ contains only approximations of the concrete solution. We choose which approximation to use according to the corresponding set quantification: In case we construct $\mathcal{A}_i$ and $\mathbb{Q}_i = \exists$, we use the *underapproximation* (thus making sure that any witness trace we pick is indeed contained in the actual model of the second-order variable); and if $\mathbb{Q}_i = \forall$, we use the *overapproximation* (making sure that we consider at least those traces that are in the actual solution). If $\mathcal{L}(\mathcal{A}_1)$ is non-empty, i.e., accepts the empty trace assignment, the formula holds (assuming the approximations returned by `underApprox` and `overApprox` are sound). If not, we increase the precision bound N and repeat.

In Algorithm 1, we only check for the satisfaction of a formula (to keep the notation succinct). Using the second-order approximations in $\flat$ we can also check the negation of a formula (by considering the negated body and dualizing all trace quantifiers). Our tool (Section 6) makes use of this and thus simultaneously tries to show satisfaction and violation of a formula.

5.5 Computing Under- and Overapproximations

In this section we provide concrete instantiations for `underApprox` and `overApprox`.

Computing Underapproximations. As we consider the fixpoint fragment, each formula φ_j^{con} (defining Y_j) is a conjunction of formulas of the form in Equation (2), thus defining Y_j via a least fixpoint computation. For simplicity, we assume that Y_j is defined by the single conjunct, given by Equation (2) (our construction generalizes easily to a conjunction of such formulas). Assuming fixed models for $\mathfrak{S}$, $\mathfrak{A}$ and $Y_1, \ldots, Y_{j-1}$, the fixpoint operation defining Y_j is monotone, i.e., the larger the current model for Y_j is, the more traces we need to add according to Equation (2). Monotonicity allows us to apply the Knaster–Tarski theorem [45] and compute underapproximations to the fixpoint by iteration.

In our construction of an approximation for Y_j, we are given a mapping $\flat$ that fixes a pair of automata for $\mathfrak{S}$, $\mathfrak{A}$, and $Y_1, \ldots, Y_{j-1}$ (due to the outside-in traversal in Algorithm 1). As we are computing an underapproximation, we use the underapproximation for each of the second-order variables in $\flat$. So $\flat(\mathfrak{S})$ and $\flat(\mathfrak{A})$ are automata over Σ^1 and for each $j' \in [j-1]$, $\flat(Y_{j'})$ is an automaton over $\Sigma^{l_{j'}+1}$. Given this fixed mapping $\flat$, we iteratively construct automata $\hat{\mathcal{C}}_0, \hat{\mathcal{C}}_1, \ldots$ over Σ^{l_j+1} that capture (increasingly precise) underapproximations on the solution for Y_j. We set $\hat{\mathcal{C}}_0$ to be the automaton with the empty language. We then recursively define $\hat{\mathcal{C}}_{N+1}$ based on $\hat{\mathcal{C}}_N$ as follows: For each second-order variable X_i for $i \in [n]$ used in Equation (2) we can assume a concrete assignment in the form of an automaton $\mathcal{D}_i$ over Σ^{T_i+1} for some $T_i \leq l_j$: In case $X_i \neq Y_j$ (so $X_i \in \{\mathfrak{S}, \mathfrak{A}, Y_1, \ldots, Y_{j-1}\}$), we set $\mathcal{D}_i := \flat(X_i)$. In case $X_i = Y_j$, we set $\mathcal{D}_i := \hat{\mathcal{C}}_N$, i.e., we use the current approximation of Y_j in iteration N. After we have set $\mathcal{D}_1, \ldots, \mathcal{D}_n$, we compute an automaton $\dot{\mathcal{C}}$ over Σ^{l_j+1} that accepts $zip(t_1, \ldots, t_{l_j}, t)$ iff there exists traces $\dot{t}_1, \ldots, \dot{t}_n$ such that (1) $zip(t_1, \ldots, t_{T_i}, \dot{t}_i) \in \mathcal{L}(\mathcal{D}_i)$ for all $i \in [n]$, (2) $[\pi_1 \mapsto t_1, \ldots, \pi_{l_j} \mapsto t_{l_j}, \dot{\pi}_1 \mapsto \dot{t}_1, \ldots, \dot{\pi}_n \mapsto \dot{t}_n] \vDash \psi_{step}$, and (3) trace t equals $\dot{t}_M$ (of Equation (2)). The intuition is that $\dot{\mathcal{C}}$ captures all traces that should be added to Y_j: Given $t_1, \ldots, t_{l_j}$ we check if there are traces $\dot{t}_1, \ldots, \dot{t}_n$ for trace variables $\dot{\pi}_1, \ldots, \dot{\pi}_n$ in Equation (2) where (1) each $\dot{t}_i$ is in the assignment for X_i, which is captured by the automaton $\mathcal{D}_i$ over Σ^{T_i+1}, and (2) the traces $\dot{t}_1, \ldots, \dot{t}_n$ satisfy φ_{step}. If this is the case, we want to add $\dot{t}_M$ (as stated in Equation (2)). We then define $\hat{\mathcal{C}}_{N+1}$ as the union of $\hat{\mathcal{C}}_N$ and $\dot{\mathcal{C}}$, i.e. extend the previous model with all (potentially new) traces that need to be added.

Computing Overapproximations. As we noted above, conditions of the form of Equation (2) always define fixpoint constraints. To compute upper bounds on such fixpoint constructions we make use of Park's theorem, [46] stating that if we find some set (or automaton) $\mathcal{B}$ that is inductive (i.e., when computing all traces that we would need to add assuming the current model of Y_j is $\mathcal{B}$, we end up with traces that are already in $\mathcal{B}$), then $\mathcal{B}$ overapproximates the unique solution (aka. least fixpoint) of Y_j. To derive such an inductive invariant, we employ techniques developed in the context of regular model checking [14] (see Section 7). Concretely, we employ the approach from [18] that uses automata learning [2] to find suitable invariants. While the approach from [18] is limited to finite words, we extend it to an ω-setting by interpreting an automaton accepting finite words as one that accepts an ω-word u iff every prefix of u is accepted.[2] As soon as the learner provides a candidate for an equivalence check, we check that it is inductive and, if not, provide some finite counterexample (see [18] for details). If the automaton is inductive, we return it as a potential overapproximation. Should this approximation not be precise enough, the first-order model checking (Section 5.3) returns some concrete counterexample, i.e., some trace contained in the invariant but violating the property, which we use to provide more counterexamples to the learner.

6 Implementation and Experiments

We have implemented our model-checking algorithm in a prototype tool we call `HySO` (**Hy**perproperties with **S**econd **O**rder).[3] Our tool uses `spot` [28] for basic

[2] This effectively poses the assumption that the step formula specifies a safety property, which seems to be the case for almost all examples. As an example, common knowledge infers a safety property: In each step, we add all traces for which there exists some trace that agrees on all propositions observed by that agent.

[3] Our tool is publicly available at https://doi.org/10.5281/zenodo.7877144.

automata operations (such as LTL-to-NBA translations and complementations). To compute under- and overapproximations, we use the techniques described in Section 5.5. We evaluate the algorithm on the following benchmarks.

Muddy Children. The muddy children puzzle [29] is one of the classic examples in common knowledge literature. The puzzle consists of n children standing such that each child can see all other children's faces. From the n children, an unknown number $k \geq 1$ have a muddy forehead, and in incremental rounds, the children should step forward if they know if their face is muddy or not. Consider the scenario of $n = 2$ and $k = 1$, so child a sees that child b has a muddy forehead and child b sees that a is clean. In this case, b immediately steps forward, as it knows that its forehead is muddy since $k \geq 1$. In the next step, a knows that its face is clean since b stepped forward in round 1. In general, one can prove that all children step forward in round k, deriving common knowledge.

For each n we construct a transition system $\mathcal{T}_n$ that encodes the muddy children scenario with n children. For every m we design a Hyper^{2}LTL$_{\text{fp}}$ formula φ_m that adds to the common knowledge set X all traces that appear indistinguishable in the first m steps for some child. We then specify that all traces in X should agree on all inputs, asserting that all inputs are common knowledge.[4] We used `HySO` to *fully automatically* check $\mathcal{T}_n$ against φ_m for varying values of n and m, i.e., we checked if, after the first m steps, the inputs of all children are common knowledge. As expected, the above property holds only if $m \geq n$ (in the worst case, where all children are dirty ($k = n$), the inputs of all children only become common knowledge after n steps). We depict the results in Table 1a.

Asynchronous Hyperproperties. As we have shown in Section 4.2, we can encode arbitrary AHLTL properties into Hyper^{2}LTL$_{\text{fp}}$. We verified synchronous and asynchronous version of observational determinism (cf. Section 4.2) on programs taken from [3,5,9]. We depict the verification results in Table 1b. Recall that Hyper^{2}LTL$_{\text{fp}}$ properties without any second-order variables correspond to HyperQPTL formulas. `HySO` can check such properties precisely, i.e., it constitutes a sound-and-complete model checker for HyperQPTL properties with an arbitrary quantifier prefix. The synchronous version of observational determinism is a HyperLTL property and thus needs no second-order approximation (we set the method column to "-" in these cases).

Common Knowledge in Multi-agent Systems. We used `HySO` for an automatic analysis of the system in Figure 1. Here, we verify that on initial trace $\{a\}^n\{d\}^\omega$ it is CK that a holds in the first step. We use a similar formula as

[4] This property is not expressible in non-hyper logics such as LTL$_{\mathsf{K,C}}$, where we can only check *trace properties* on the common knowledge set X. In contrast, Hyper^{2}LTL$_{\text{fp}}$ allows us to check *hyperproperties* on X. That way, we can express that some value is common knowledge (i.e., equal across all traces in the set) and not only that a property is common knowledge (i.e., holds on all traces in the set).

n \ m	1	2	3	4
2	✗ 0.64	✓ 0.59		
3	✗ 0.79	✗ 0.75	✓ 0.54	
4	✗ 2.72	✗ 2.21	✗ 1.67	✓ 1.19

(a)

Instance	**Method**	**Res**	***t***
$\mathcal{T}_{syn}$, φ_{OD}	-	✓	0.26
$\mathcal{T}_{asyn}$, φ_{OD}	-	✗	0.31
$\mathcal{T}_{syn}$, φ_{OD}^{asyn}	Iter (0)	✓	0.50
$\mathcal{T}_{syn}$, φ_{OD}^{asyn}	Iter (1)	✓	0.78
Q1, φ_{OD}	-	✗	0.34
Q1, φ_{OD}^{asyn}	Iter (1)	✓	0.86

(b)

Table 1: In Table 1a, we check common knowledge in the muddy children puzzle for n children and m rounds. We give the result (✓ if common knowledge holds and ✗ if it does not), and the running time. In Table 1b, we check synchronous and asynchronous versions of observational determinism. We depict the number of iterations needed and running time. Times are given in seconds.

the one of Section 3.3, with the change that we are interested in whether a is CK (whereas we used $\bigcirc a$ in Section 3.3). As expected, `HySO` requires $2n-1$ iterations to converge. We depict the results in Table 2a.

Mazurkiewicz Traces. Mazurkiewicz traces are an important concept in the theory of distributed computing [26]. Let $I \subseteq \Sigma \times \Sigma$ be an independence relation that determines when two consecutive letters can be switched (think of two actions in disjoint processes in a distributed system). Any $t \in \Sigma^\omega$ then defines the set of all traces that are equivalent to t by flipping consecutive independent actions an arbitrary number of times (the equivalence class of all these traces is called the Mazurkiewicz Trace). See [26] for details. The verification problem for Mazurkiewicz traces now asks if, given some $t \in \Sigma^\omega$, all traces in the Mazurkiewicz trace of t satisfy some property ψ. Using Hyper^{2}LTL$_{\text{fp}}$ we can directly reason about the Mazurkiewicz Trace of any given trace, by requiring that all traces that are equal up to one swap of independent letters are also in a given set (which is easily expressed in Hyper^{2}LTL$_{\text{fp}}$).

Using `HySO` we verify a selection of such trace properties that often require non-trivial reasoning by coming up with a suitable invariant. We depict the results in Table 2b. In our preliminary experiments, we model a situation where we start with $\{a\}^1\{\}^\omega$ and can swap letters $\{a\}$ and $\{\}$. We then, e.g., ask if on any trace in the resulting Mazurkiewicz trace, a holds at most once, which requires inductive invariants and cannot be established by iteration.

n	Method	Res	t
1	Iter (1)	✓	0.51
2	Iter (3)	✓	0.83
3	Iter (5)	✓	1.20
10	Iter (19)	✓	3.81
100	Iter (199)	✓	102.8

(a)

Instance	Method	Res	t
SwapA	Learn	✓	1.07
SwapATwice	Learn	✓	2.13
SwapA$_5$	Iter (5)	✓	1.15
SwapA$_{15}$	Iter (15)	✓	3.04
SwapAViolation$_5$	Iter (5)	✗	2.35
SwapAViolation$_{15}$	Iter (15)	✗	4.21

(b)

Table 2: In Table 1a, we check common knowledge in the example from Figure 1 when starting with $a^n d^\omega$ for varying values of n. We depict the number of refinement iterations, the result, and the running time. In Table 2b, we verify various properties on Mazurkiewicz traces. We depict whether the property could be verified or refuted by iteration or automata learning, the result, and the time. Times are given in seconds.

7 Related Work

In recent years, many logics for the formal specification of hyperproperties have been developed, extending temporal logics with explicit path quantification (examples include HyperLTL, HyperCTL* [19], HyperQPTL [43,10], HyperPDL [36], and HyperATL* [5,9]); extending first and second-order logics with an equal level predicate [24,31]; or extending (ω)-regular hyperproperties [35,13] to context-free hyperproperties [33]. Hyper^{2}LTL is the first temporal logic that reasons about second-order hyperproperties which allows is to capture many existing (epistemic, asynchronous, etc.) hyperlogics while at the same time taking advantage of model-checking solutions that have been proven successful in first-order settings.

Asynchronous Hyperproperties. For asynchronous hyperproperties, Gutfeld et al. [37] present an asynchronous extension of the polyadic μ-calculus. Bozelli et al. [16] extend HyperLTL with temporal operators that are only evaluated if the truth value of some temporal formula changes. Baumeister et al. present AHLTL [3], that extends HyperLTL with a explicit quantification over trajectories and can be directly encoded within Hyper^{2}LTL$_{\text{fp}}$.

Regular Model Checking. Regular model checking [14] is a general verification method for (possibly infinite state) systems, in which each state of the system is interpreted as a finite word. The transitions of the system are given as a finite-state (regular) transducer, and the model checking problem asks if, from some initial set of states (given as a regular language), some bad state is eventually reachable. Many methods for automated regular model checking have been developed [11,25,12,18]. Hyper^{2}LTL can be seen as a logical foundation for ω-regular model checking: Assume the set of initial states is given as a QPTL formula φ_{init}, the set of bad states is given as a QPTL formula φ_{bad}, and the transition relation is given as a QPTL formula φ_{step} over trace variables π and π'. The set of bad states is reachable from a trace (state) in φ_{init} iff the following Hyper^{2}LTL$_{\mathrm{fp}}$ formula holds on the system that generates all traces:

$$\big(X, \curlyvee, \forall \pi \in \mathfrak{S}.\, \varphi_{init}(\pi) \rightarrow \pi \triangleright X \wedge \\ \forall \pi \in X. \forall \pi' \in \mathfrak{S}.\, \varphi_{step}(\pi, \pi') \rightarrow \pi' \triangleright X\big).\, \forall \pi \in X. \neg \varphi_{bad}(\pi)$$

Conversely, Hyper^{2}LTL$_{\mathrm{fp}}$ can express more complex properties, beyond the reachability checks possible in the framework of (ω-)regular model checking.

Model Checking Knowledge. Model checking of knowledge properties in multi-agent systems was developed in the tools `MCK` [34] and `MCMAS` [40], which can exactly express LTL$_{\mathsf{K}}$. Bozzelli et al. [15] have shown that HyperCTL* and LTL$_{\mathsf{K}}$ have incomparable expressiveness, and present HyperCTL$^*_{lp}$ – an extension of HyperCTL* that can reason about past – to unify HyperCTL* and LTL$_{\mathsf{K}}$. While HyperCTL$^*_{lp}$ can express the knowledge operator, it cannot capture common knowledge. LTL$_{\mathsf{K},\mathsf{C}}$ [39] captures both knowledge and common knowledge, but the suggested model-checking algorithm only handles a decidable fragment that is reducible to LTL model checking.

8 Conclusion

Hyperproperties play an increasingly important role in many areas of computer science. There is a strong need for specification languages and verification methods that reason about hyperproperties in a uniform and general manner, similar to what is standard for more traditional notions of safety and reliability. In this paper, we have ventured forward from the first-order reasoning of logics like HyperLTL into the realm of second-order hyperproperties, i.e., properties that not only compare individual traces but reason comprehensively about *sets* of such traces. With Hyper^{2}LTL, we have introduced a natural specification language and a general model-checking approach for second-order hyperproperties. Hyper^{2}LTL provides a general framework for a wide range of relevant hyperproperties, including common knowledge and asynchronous hyperproperties, which could previously only be studied with specialized logics and algorithms. Hyper^{2}LTL also provides a starting point for future work on second-order hyperproperties in areas such as cyber-physical [42] and probabilistic systems [27].

Acknowledgements. We thank Jana Hofmann for the fruitful discussions. This work was supported by the European Research Council (ERC) Grant HYPER (No. 101055412), by DFG grant 389792660 as part of TRR 248, and by the German Israeli Foundation (GIF) Grant No. I-1513-407.2019.

References

1. Alur, R., Henzinger, T.A.: A really temporal logic. J. ACM **41**(1) (1994). https://doi.org/10.1145/174644.174651
2. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2) (1987). https://doi.org/10.1016/0890-5401(87)90052-6
3. Baumeister, J., Coenen, N., Bonakdarpour, B., Finkbeiner, B., Sánchez, C.: A temporal logic for asynchronous hyperproperties. In: International Conference on Computer Aided Verification, CAV 2021. Lecture Notes in Computer Science, vol. 12759. Springer (2021). https://doi.org/10.1007/978-3-030-81685-8_33
4. Beutner, R., Carral, D., Finkbeiner, B., Hofmann, J., Krötzsch, M.: Deciding hyperproperties combined with functional specifications. In: Annual ACM/IEEE Symposium on Logic in Computer, LICS 2022. ACM (2022). https://doi.org/10.1145/3531130.3533369
5. Beutner, R., Finkbeiner, B.: A temporal logic for strategic hyperproperties. In: International Conference on Concurrency Theory, CONCUR 2021. LIPIcs, vol. 203. Schloss Dagstuhl (2021). https://doi.org/10.4230/LIPIcs.CONCUR.2021.24
6. Beutner, R., Finkbeiner, B.: Prophecy variables for hyperproperty verification. In: IEEE Computer Security Foundations Symposium, CSF 2022. IEEE (2022). https://doi.org/10.1109/CSF54842.2022.9919658
7. Beutner, R., Finkbeiner, B.: Software verification of hyperproperties beyond k-safety. In: International Conference on Computer Aided Verification, CAV 2022. Lecture Notes in Computer Science, vol. 13371. Springer (2022). https://doi.org/10.1007/978-3-031-13185-1_17
8. Beutner, R., Finkbeiner, B.: AutoHyper: Explicit-state model checking for HyperLTL. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2023. vol. 13993. Springer (2023). https://doi.org/10.1007/978-3-031-30823-9_8
9. Beutner, R., Finkbeiner, B.: HyperATL*: A logic for hyperproperties in multi-agent systems. Log. Methods Comput. Sci. (2023)
10. Beutner, R., Finkbeiner, B.: Model checking omega-regular hyperproperties with AutoHyperQ. In: International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2023. EPiC Series in Computing, EasyChair (2023)
11. Boigelot, B., Legay, A., Wolper, P.: Iterating transducers in the large. In: International Conference on Computer Aided Verification, CAV 2003. Lecture Notes in Computer Science, vol. 2725. Springer (2003). https://doi.org/10.1007/978-3-540-45069-6_24
12. Boigelot, B., Legay, A., Wolper, P.: Omega-regular model checking. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2004. Lecture Notes in Computer Science, vol. 2988. Springer (2004). https://doi.org/10.1007/978-3-540-24730-2_41
13. Bonakdarpour, B., Sheinvald, S.: Finite-word hyperlanguages. In: Leporati, A., Martín-Vide, C., Shapira, D., Zandron, C. (eds.) Language and Automata Theory and Applications - 15th International Conference, LATA 2021, Milan, Italy, March 1-5, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12638, pp. 173–186. Springer (2021). https://doi.org/10.1007/978-3-030-68195-1_17, `https://doi.org/10.1007/978-3-030-68195-1_17`
14. Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: International Conference on Computer Aided Verification, CAV

2000. Lecture Notes in Computer Science, vol. 1855. Springer (2000). https://doi.org/10.1007/10722167_31

15. Bozzelli, L., Maubert, B., Pinchinat, S.: Unifying hyper and epistemic temporal logics. In: International Conference on Foundations of Software Science and Computation Structures, FoSSaCS 2015. Lecture Notes in Computer Science, vol. 9034. Springer (2015). https://doi.org/10.1007/978-3-662-46678-0_11
16. Bozzelli, L., Peron, A., Sánchez, C.: Asynchronous extensions of HyperLTL. In: Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021. IEEE (2021). https://doi.org/10.1109/LICS52264.2021.9470583
17. Büchi, J.R.: On a decision method in restricted second-order arithmetic. In: Studies in Logic and the Foundations of Mathematics. vol. 44. Elsevier (1966)
18. Chen, Y., Hong, C., Lin, A.W., Rümmer, P.: Learning to prove safety over parameterised concurrent systems. In: Formal Methods in Computer Aided Design, FMCAD 2017. IEEE (2017). https://doi.org/10.23919/FMCAD.2017.8102244
19. Clarkson, M.R., Finkbeiner, B., Koleini, M., Micinski, K.K., Rabe, M.N., Sánchez, C.: Temporal logics for hyperproperties. In: International Conference on Principles of Security and Trust, POST 2014. Lecture Notes in Computer Science, vol. 8414. Springer (2014). https://doi.org/10.1007/978-3-642-54792-8_15
20. Clarkson, M.R., Schneider, F.B.: Hyperproperties. J. Comput. Secur. **18**(6) (2010). https://doi.org/10.3233/JCS-2009-0393
21. Coenen, N., Dachselt, R., Finkbeiner, B., Frenkel, H., Hahn, C., Horak, T., Metzger, N., Siber, J.: Explaining hyperproperty violations. In: International Conference on Computer Aided Verification, CAV 2022. Lecture Notes in Computer Science, vol. 13371. Springer (2022). https://doi.org/10.1007/978-3-031-13185-1_20
22. Coenen, N., Finkbeiner, B., Frenkel, H., Hahn, C., Metzger, N., Siber, J.: Temporal causality in reactive systems. In: International Symposium on Automated Technology for Verification and Analysis, ATVA 2022. Lecture Notes in Computer Science, vol. 13505. Springer (2022). https://doi.org/10.1007/978-3-031-19992-9_13
23. Coenen, N., Finkbeiner, B., Hahn, C., Hofmann, J.: The hierarchy of hyperlogics. In: Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019. IEEE (2019). https://doi.org/10.1109/LICS.2019.8785713
24. Coenen, N., Finkbeiner, B., Sánchez, C., Tentrup, L.: Verifying hyperliveness. In: International Conference on Computer Aided Verification, CAV 2019. Lecture Notes in Computer Science, vol. 11561. Springer (2019). https://doi.org/10.1007/978-3-030-25540-4_7
25. Dams, D., Lakhnech, Y., Steffen, M.: Iterating transducers. In: International Conference on Computer Aided Verification, CAV 2001. Lecture Notes in Computer Science, vol. 2102. Springer (2001). https://doi.org/10.1007/3-540-44585-4_27
26. Diekert, V., Rozenberg, G. (eds.): The Book of Traces. World Scientific (1995). https://doi.org/10.1142/2563
27. Dimitrova, R., Finkbeiner, B., Torfah, H.: Probabilistic hyperproperties of markov decision processes. In: International Symposium on Automated Technology for Verification and Analysis, ATVA 2020. Lecture Notes in Computer Science, vol. 12302. Springer (2020). https://doi.org/10.1007/978-3-030-59152-6_27
28. Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Aisse, A.G., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., Lauko, H.: From spot 2.0 to spot 2.10: What's new? In: International Conference on Computer Aided Verification, CAV 2022. Lecture Notes in Computer Science, vol. 13372. Springer (2022). https://doi.org/10.1007/978-3-031-13188-2_9
29. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. MIT Press (1995). https://doi.org/10.7551/mitpress/5803.001.0001

30. Finkbeiner, B., Rabe, M.N., Sánchez, C.: Algorithms for model checking HyperLTL and HyperCTL*. In: International Conference on Computer Aided Verification, CAV 2015. Lecture Notes in Computer Science, vol. 9206. Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_3
31. Finkbeiner, B., Zimmermann, M.: The first-order logic of hyperproperties. In: Symposium on Theoretical Aspects of Computer Science, STACS 2017. LIPIcs, vol. 66. Schloss Dagstuhl (2017). https://doi.org/10.4230/LIPIcs.STACS.2017.30
32. Fortin, M., Kuijer, L.B., Totzke, P., Zimmermann, M.: HyperLTL satisfiability is Σ_1^1-complete, HyperCTL* satisfiability is Σ_1^2-complete. In: International Symposium on Mathematical Foundations of Computer Science, MFCS 2021. LIPIcs, vol. 202. Schloss Dagstuhl (2021). https://doi.org/10.4230/LIPIcs.MFCS.2021.47
33. Frenkel, H., Sheinvald, S.: Realizable and context-free hyperlanguages. In: Ganty, P., Monica, D.D. (eds.) Proceedings of the 13th International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2022, Madrid, Spain, September 21-23, 2022. EPTCS, vol. 370, pp. 114–130 (2022). https://doi.org/10.4204/EPTCS.370.8, `https://doi.org/10.4204/EPTCS.370.8`
34. Gammie, P., van der Meyden, R.: MCK: model checking the logic of knowledge. In: International Conference on Computer Aided Verification, CAV 2004. Lecture Notes in Computer Science, vol. 3114. Springer (2004). https://doi.org/10.1007/978-3-540-27813-9_41
35. Goudsmid, O., Grumberg, O., Sheinvald, S.: Compositional model checking for multi-properties. In: Henglein, F., Shoham, S., Vizel, Y. (eds.) Verification, Model Checking, and Abstract Interpretation - 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17-19, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12597, pp. 55–80. Springer (2021). https://doi.org/10.1007/978-3-030-67067-2_4, `https://doi.org/10.1007/978-3-030-67067-2_4`
36. Gutsfeld, J.O., Müller-Olm, M., Ohrem, C.: Propositional dynamic logic for hyperproperties. In: International Conference on Concurrency Theory, CONCUR 2020. LIPIcs, vol. 171. Schloss Dagstuhl (2020). https://doi.org/10.4230/LIPIcs.CONCUR.2020.50
37. Gutsfeld, J.O., Müller-Olm, M., Ohrem, C.: Automata and fixpoints for asynchronous hyperproperties. Proc. ACM Program. Lang. **5**(POPL) (2021). https://doi.org/10.1145/3434319
38. Halpern, J.Y., Moses, Y.: Knowledge and common knowledge in a distributed environment. J. ACM **37**(3), 549–587 (1990)
39. van der Hoek, W., Wooldridge, M.J.: Model checking knowledge and time. In: International Workshop on Model Checking of Software, SPIN 2002. Lecture Notes in Computer Science, vol. 2318. Springer (2002). https://doi.org/10.1007/3-540-46017-9_9
40. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: an open-source model checker for the verification of multi-agent systems. Int. J. Softw. Tools Technol. Transf. **19**(1) (2017). https://doi.org/10.1007/s10009-015-0378-x
41. van der Meyden, R.: Common knowledge and update in finite environments. Inf. Comput. **140**(2) (1998). https://doi.org/10.1006/inco.1997.2679
42. Nguyen, L.V., Kapinski, J., Jin, X., Deshmukh, J.V., Johnson, T.T.: Hyperproperties of real-valued signals. In: ACM-IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2017. ACM (2017). https://doi.org/10.1145/3127041.3127058
43. Rabe, M.N.: A temporal logic approach to information-flow control. Ph.D. thesis, Saarland University (2016)

44. Sistla, A.P.: Theoretical issues in the design and verification of distributed systems. Ph.D. thesis, Harvard University (1983)
45. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. (1955)
46. Winskel, G.: The formal semantics of programming languages - an introduction. Foundation of computing series, MIT Press (1993)

A Additional Material for Section 3

A.1 Additional Material for Section 3.1

Lemma 1. Hyper^{2}LTL *subsumes* HyperQPTL *(and thus also* HyperLTL*).*

Proof. HyperQPTL extends HyperLTL with quantification over atomic propositions. However, HyperQPTL can also be expressed using quantification over arbitrary traces, that are not necessarily system traces, capturing exactly the same semantics. Then, we can translate every HyperLTL trace quantification $\mathbb{Q}\pi.\varphi$ to $\mathbb{Q}\pi \in \mathfrak{S}.\varphi$, and every HyperQPTL trace quantification $\mathbb{Q}\tau.\varphi$ to $\mathbb{Q}\tau \in \mathfrak{A}.\varphi$, to obtain a Hyper^{2}LTL formula. □

A.2 Additional Material for Section 3.2

Proposition 1. *Any* Hyper^{2}LTL$_{\text{fp}}$ *formula* φ *can be effectively translated into an* Hyper^{2}LTL *formula* φ' *such that for all transition systems* $\mathcal{T}$ *we have* $\mathcal{T} \models \varphi$ *iff* $\mathcal{T} \models \varphi'$*.*

Proof. We translate the Hyper^{2}LTL$_{\text{fp}}$ formula φ into a Hyper^{2}LTL formula $\llbracket\varphi\rrbracket$:

$$\begin{aligned}
\llbracket\psi\rrbracket &:= \psi\\
\llbracket\mathbb{Q}\pi \in X.\, \varphi\rrbracket &:= \mathbb{Q}\pi \in X.\, \llbracket\varphi\rrbracket\\
\llbracket\exists(X, \curlyvee, \varphi_1).\, \varphi_2\rrbracket &:= \exists X.\, \llbracket\varphi_1\rrbracket \land \big(\forall Y.\, Y \subsetneq X \Rightarrow \neg\llbracket\varphi_1[Y/X]\rrbracket\big) \land \llbracket\varphi_2\rrbracket\\
\llbracket\forall(X, \curlyvee, \varphi_1).\, \varphi_2\rrbracket &:= \forall X.\, \Big(\llbracket\varphi_1\rrbracket \land \big(\forall Y.\, Y \subsetneq X \Rightarrow \neg\llbracket\varphi_1[Y/X]\rrbracket\big)\Big) \Rightarrow \llbracket\varphi_2\rrbracket\\
\llbracket\exists(X, \curlywedge, \varphi_1).\, \varphi_2\rrbracket &:= \exists X.\, \llbracket\varphi_1\rrbracket \land \big(\forall Y.\, Y \supsetneq X \Rightarrow \neg\llbracket\varphi_1[Y/X]\rrbracket\big) \land \llbracket\varphi_2\rrbracket\\
\llbracket\forall(X, \curlywedge, \varphi_1).\, \varphi_2\rrbracket &:= \forall X.\, \Big(\llbracket\varphi_1\rrbracket \land \big(\forall Y.\, Y \supsetneq X \Rightarrow \neg\llbracket\varphi_1[Y/X]\rrbracket\big)\Big) \Rightarrow \llbracket\varphi_2\rrbracket
\end{aligned}$$

Path formulas and first-order quantification can be translated verbatim. To translate (fixpoint-based) second-order quantification we use additional second order quantification to express the fact that a set should be a least or greatest. We write $Y \subsetneq X$ as a shorthand for

$$\big(\forall \pi \in Y. \pi \triangleright X\big) \land \big(\exists \pi \in Y. \forall \pi' \in Y. \Diamond \neg(\pi =_{\text{AP}} \pi')\big)$$

$\varphi_1[Y/X]$ denotes the formula where all free occurrences of X are replaced by Y.

Note that above formula is no Hyper^{2}LTL formula as it is not in prenex normal form. However, no (first or second-order) quantification occurs under temporal operators, so we can easiliy bring it into prenex normal form. It is easy to see that any system satisfies φ in the Hyper^{2}LTL$_{\text{fp}}$ semantics iff it satisfies $\llbracket\varphi\rrbracket$ in the Hyper^{2}LTL semantics. □

A.3 Additional Proofs for Section 3.4

Proposition 3. *The model-checking problem of* Hyper^{2}LTL$_{\text{fp}}$ *is* Σ_1^1*-hard.*

Proof. Instead of working with Turing machines, we consider two counter machines as they are simpler to handle. A two-counter machine (2CM) maintains two counters c_1, c_2 and has a finite set of instructions $l_1, \dots, l_n$. Each instruction l_i is of one of the following forms, where $x \in \{1, 2\}$ and $1 \leq j, k \leq n$.

- $l_i : \big[c_x \coloneqq c_x + 1;\ \texttt{goto}\ \{l_j, l_k\}\big]$
- $l_i : \big[c_x \coloneqq c_x - 1;\ \texttt{goto}\ \{l_j, l_k\}\big]$
- $l_i : \big[\texttt{if}\ c_x = 0\ \texttt{then goto}\ l_j\ \texttt{else goto}\ l_k\big]$

Here, $\texttt{goto}\ \{l_j, l_k\}$ indicates that the machine nondeterministically chooses between instructions l_j and l_k. A configuration of a 2CM is a tuple (l_i, v_1, v_2), where l_i is the next instruction to be executed, and $v_1, v_2 \in \mathbb{N}$ denote the values of the counters. The initial configuration of a 2CM is $(l_1, 0, 0)$. The transition relation between configurations is defined as expected. Decrementing a counter that is already 0 leaves the counter unchanged. An infinite computation is *recurring* if it visits instruction l_1 infinitely many times. Deciding if a machine has a recurring computation is Σ_1^1-hard [1].

For our proof, we encode each configuration of the 2CM as an infinite trace. We use to atomic propositions c_1, c_2, which we ensure to hold exactly once and use this unique position to represent the counter value. The current instruction can be encoded in the first position using APs $l_1, \dots, l_n$. We further use a fresh AP $\dagger$ – which also holds exactly once – to mark the step of this configuration. We are interested in a set of traces X that satisfies all of the following requirements:

1. The set X contains the initial configuration. Note that in this configuration, $\dagger$ holds in the *first* step.
2. For every configuration, there exists a successor configuration. Note that in the successor configuration, $\dagger$ is shifted by one position.
3. All pairs of traces where $\dagger$ holds at the same position are equal. X thus assigns a unique configuration to each step.
4. The computation is recurrent. As already done in [4], we can ensure this by adding a fresh counter that counts down to the next visit of instruction l_1.

It is easy to see that we can encode all the above as a Hyper^{2}LTL (or Hyper^{2}LTL$_{\text{fp}}$) formula using only first-order quantification over traces in X. Let φ be such a formula. It is easy to see that the Hyper^{2}LTL formula $\exists X. \varphi$ holds on any system, iff there exists a set X with the above properties iff the 2CM has a recurring computation. The reproves Proposition 2.

For the present proof, we do, however, want to show Σ_1^1-hardness for the less powerful Hyper^{2}LTL$_{\text{fp}}$. The key point to extend this is to ensure that iff there exists a set X that satisfies the above requirements, then there also exists a minimal one. The key observation is that – by the construction of X – any set X that satisfies the above *is already minimal*: The AP $\dagger$ ensures that, for each step, there exists exactly one configuration (Item 3), and, when removing any number of traces from X, we will inevitably violate Item 4.

We thus get that the 2CM has a recurring computation iff there exists *mininmal* X that satisfies φ iff the Hyper^{2}LTL$_{\text{fp}}$ formula $\exists(X, \curlyvee, \varphi).\, \mathit{true}$ holds on an arbitrary system. □

Proposition 4. *Let φ be a* Hyper^{2}LTL *formula that uses only existential second-order quantification and $\mathcal{T}$ be any system. We can effectively construct a formula φ' in* HyperQPTL *such that $\mathcal{T} \vDash \varphi$ iff φ' is satisfiable.*

Proof. Assume that we have m second-order quantifiers, and for each $k \in [m]$, $\pi_1, \dots, \pi_{l_k}$ are the first-order variables occurring before X_k is quantified:

$$\varphi = \mathbb{Q}\pi_1, \dots, \mathbb{Q}\pi_{l_1} \exists X_1 \mathbb{Q}\pi_{l_1+1}, \dots, \mathbb{Q}\pi_{l_2} \exists X_2 \cdots \exists X_m \mathbb{Q}\pi_{l_m+1}, \dots, \mathbb{Q}\pi_{l_{m+1}} \psi \ ,$$

The second-order variables we use thus are $\mathfrak{V} = \{\mathfrak{S}, \mathfrak{A}, X_1, \dots, X_m\}$. Each $X \in \mathfrak{V}$ can depend on some of the traces quantified before it. In particular, each X_k depends on traces $\pi_1, \dots, \pi_{l_k}$, $\mathfrak{S}$ depends on none of the traces (as it is fixed) and neither does $\mathfrak{A}$. We define a function $c : \mathfrak{V} \to \mathbb{N}$ that denotes on how many traces the set can depend, i.e., $c(\mathfrak{S}) = c(\mathfrak{A}) = 0$, and $c(X_k) = l_k$.

We then encode this functional dependence into a model by using traces. For each $X \in \mathfrak{V}$, we define atomic propositions

$$\text{AP}_X := \{[a, j]_X \mid a \in \text{AP} \land j \in [c(X)] \cup \{\dagger\}\}$$

and then define

$$\text{AP}' := \text{AP} \uplus \bigcup_{X \in \mathfrak{V}} \text{AP}_X$$

We will use the original APs to describe traces. The additional propositions are used to encode functions which map the $c(X)$ traces quantified before X to some set of traces. Given traces $\pi_1, \dots, \pi_{C(X)}$, we say a trace t is in the model of X if there exists some trace $\dot{t}$ (in the model of our final formula) such that

$$\begin{aligned} \forall z \in \mathbb{N}. \Big(\bigwedge_{j \in [C(X)]} \bigwedge_{a \in \text{AP}} \big(a \in t_j(z) \leftrightarrow [a, j]_X \in \dot{t}(z)\big)\Big) \land \\ \forall z \in \mathbb{N}. \Big(\bigwedge_{a \in \text{AP}} \big(a \in t(z) \leftrightarrow [a, \dagger]_X \in \dot{t}(z)\big)\Big) \end{aligned} \tag{3}$$

holds. That is, the trace $\dot{t}$ defines the functional mapping from $t_1, \dots, t_{c(X)}$ to t.

We use this idea of encoding functions to translate φ as follows:

$$\begin{aligned} [\![\psi]\!] &:= \psi \\ [\![\mathbb{Q}X.\varphi]\!] &:= [\![\varphi]\!] \\ [\![\exists \pi_i \in X.\, \varphi]\!] &:= \exists \pi_i.\, \langle \pi_i \in X \rangle \land [\![\varphi]\!] \\ [\![\forall \pi_i \in X.\, \varphi]\!] &:= \forall \pi_i.\, \langle \pi_i \in X \rangle \to [\![\varphi]\!] \end{aligned}$$

We leave path formulas unchanged and fully ignore second-order quantification (which is always existential). We define $\langle \pi \in X \rangle$ as an abbreviation for

$$\exists \dot{\pi}. \Big(\bigwedge_{j \in [c(X)]} \square \bigwedge_{a \in \mathrm{AP}} \big(a_{\pi_j} \leftrightarrow ([a,j]_X)_{\dot{\pi}}\big)\Big) \wedge \Big(\square \bigwedge_{a \in \mathrm{AP}} \big(a_{\pi} \leftrightarrow ([a,\dagger]_X)_{\dot{\pi}}\big)\Big)$$

which encodes Equation (3).

The last thing we need to ensure is that $\mathfrak{S}$ and $\mathfrak{A}$ are encoded correctly. That is for any trace t we have $t \in \mathit{Traces}(\mathcal{T})$ iff there exists a $\dot{t}$ (in the model) such that for any $z \in \mathbb{N}$ and $a \in \mathrm{AP}$, we have $a \in t(z)$ iff $[a,\dagger]_{\mathfrak{S}} \in \dot{t}(z)$. Similarly, there should exists a trace $\dot{t}$ (in the model) such that for any $z \in \mathbb{N}$ and $a \in \mathrm{AP}$, $[a,\dagger]_{\mathfrak{A}} \in \dot{t}(z)$. Both requirement can be easily expressed as HyperQPTL formulas $\varphi_{\mathfrak{S}}$ and $\varphi_{\mathfrak{A}}$ (Note that expressing these requirement in HyperLTL is not possible as we cannot quantify over traces that are not within the current model).

It is easy to see that $\mathcal{T} \models \varphi$ iff $[\![\varphi]\!] \wedge \varphi_{\mathfrak{S}} \wedge \varphi_{\mathfrak{A}}$ is satisfiable. □

Proposition 5. *The model-checking problem of* Hyper^{2}LTL *is decidable for the fragments:* $\exists^* X \forall^* \pi$, $\forall^* X \forall^* \pi$, $\exists^* X \exists^* \pi$, $\forall^* X \exists^* \pi$, $\exists X. \exists^* \pi \in X \forall^* \pi' \in X$.

Proof. $\forall^* X \forall^* \pi.\varphi$, $\exists^* X \exists^* \pi.\varphi$: As universal properties are downwards closed and existential properties are upwards closed, removing second order quantification does not change the semantics of the formula.

$\exists^* X \forall^* \pi.\varphi$: for every variable X we introduce a trace variable π_x which is existentially bounded, and for every occurrence of π in φ such that $\pi \in X$, we replace π with π_x. If φ holds for all traces in X, it holds also when replacing X with the singleton π_x. The other direction of implication is trivial as we found a set $X = \{\pi_X\}$ for which φ holds. For similar reasons, for $\forall^* X \exists^* \pi.\varphi$ we can remove the second order quantification and replace every existentially trace quantification with a universal trace quantification.

As a conclusion of all of the above, we have that the model-checking of Hyper^{2}LTLformulas of the following type is decidable: $\mathbb{Q}_1 X_1 \cdots \mathbb{Q}_k X_k . \mathbb{Q}'_1 \pi_1 \in X_1 \cdots \mathbb{Q}'_k \pi_k \in X_k.\varphi$ where we have $\mathbb{Q}_i, \mathbb{Q}'_i \in \{\exists, \forall\}$ and in φ only traces from the same set X_i are compared to each other (that is, φ does not bind traces from different sets to each other).

Lastly, for $\psi = \exists X. \exists^* \pi \in X. \forall^* \pi' \in X.\varphi$, we use a reduction to the satisfiability problem of HyperQPTL [23]. Let $\varphi_{\mathcal{T}}$ a QPTL formula that models the system. Then, $\mathcal{T} \models \psi$ iff the HyperQPTL formula $\hat{\psi}$ is satisfiable, where $\hat{\psi} = \exists^* \pi . \forall^* \pi' . \forall \tau . \varphi_{\mathcal{T}}(\tau) \wedge \psi(\pi, \pi')$. Since $\hat{\psi}$ is a $\exists^*\forall^*$ HyperQPTL formula, its satisfiability problem is decidable [23]. □

B Appendix for Section 4

B.1 Additional Material for Section 4.1

LTL$_{\mathsf{K,C}}$ is defined by the following grammar:

$$\psi := a \mid \neg\psi \mid \bigcirc \psi \mid \psi_1 \,\mathcal{U}\, \psi_2 \mid \mathsf{K}_A \psi \mid \mathsf{E}\psi \mid \mathsf{C}\psi$$

where A is a set of agents. Given two traces t, t' we write $t[0,i] =_{A_i} t'[0,i]$ if t and t' appear indistinguishable for agent A_i in the first i steps. Given a set of traces T and a trace t we define

$$
\begin{array}{lll}
t, i \vDash_T a & \text{iff} & a \in t(i) \\
t, i \vDash_T \neg\psi & \text{iff} & t, i \nvDash_T \psi \\
t, i \vDash_T \psi_1 \wedge \psi_2 & \text{iff} & t, i \vDash_T \psi_1 \text{ and } t, t \vDash_T \psi_2 \\
t, i \vDash_T \bigcirc\psi & \text{iff} & t, i+1 \vDash_T \psi \\
t, i \vDash_T \psi_1 \mathcal{U} \psi_2 & \text{iff} & \exists j \geq i.\, t, j \vDash_T \psi_2 \text{ and } \forall i \leq k < j.\, t, k \vDash_T \psi_1 \\
t, i \vDash_T \mathsf{K}_{A_i}\psi & \text{iff} & \forall t' \in T.\, t[0,i] =_{A_i} t'[0,i] \rightarrow t', i \vDash \psi \\
t, i \vDash_T \mathsf{E}\psi & \text{iff} & t, i \vDash_T \bigwedge_{A_i \in A} \mathsf{K}_{A_i}\psi \\
t, i \vDash_T \mathsf{C}\psi & \text{iff} & t, i \vDash_T \mathsf{E}^{\infty}\psi
\end{array}
$$

The *everyone knows* operator E states that every agents knows that ψ holds. The semantics of the common knowledge operator C is then the infinite chain, or transitive closure, of *everyone knows that everyone knows that ... ψ*.

Proposition 6. *For every* $\text{LTL}_{\mathsf{K},\mathsf{C}}$ *formula φ there exists an* $\text{Hyper}^2\text{LTL}_{\text{fp}}$ *formula φ' such that for any system $\mathcal{T}$ we have $\mathcal{T} \vDash_{LTL_{\mathsf{K},\mathsf{C}}} \varphi$ iff $\mathcal{T} \vDash \varphi'$.*

Proof. Let $\{A_1, \dots, A_n\}$ be the set of agents. For the jth occurrence of a knowledge operator $\mathbb{K} \in \{\mathsf{K}, \mathsf{C}\}$ we introduce a new trace variable τ_j and a second order variable Y_j. In addition, we introduce a new atomic proposition k. We then replace the jth occurrence of $\mathbb{K}$ in φ with k_{τ_j}, resulting in the HyperLTL formula ψ^{τ}. Denote by ψ_j the subformula of ψ^{τ} that directly follows k_{τ_j}.

We define by induction on the nested knowledge operators the corresponding $\text{Hyper}^2\text{LTL}_{\text{fp}}$ formula φ_j. For the first (inner-most) operator, we define φ_0 to be the LTL formula nested under this operator. Now, assume we have defined φ_{j-1}, and let $\mathbb{K} \in \{\mathsf{K}, \mathsf{C}\}$ be the jth inner-most knowledge operator. Then, φ_j is defined as follows.

$$
\forall \tau_j \in \mathfrak{A}. \Big(Y_j, \curlyvee, (\pi \in Y_j \wedge \forall \pi_1 \in Y_j . \forall \pi_2 \in \mathfrak{S}.\, \big(\neg k_{\tau_j} \,\mathcal{U} (k_{\tau_j} \wedge \bigcirc\Box\neg k_{\tau_j})\big) \rightarrow \\
\Big(equiv^j_{\mathbb{K}}(\pi_1, \pi_2) \,\mathcal{U}\, k_{\tau_j} \rightarrow \pi_2 \triangleright Y_j \Big) \Big) . \forall \pi_1 \in Y_j . \varphi_{j-1} \wedge \Box \big(k_{\tau_j} \rightarrow \psi_j\big)
$$

Where

$$
equiv^j_{\mathsf{K}_{A_i}} := \pi_1 \leftrightarrow_{A_i} \pi_2 \qquad equiv^j_{\mathsf{C}} := \bigvee_{i \in [n]} \pi_1 \leftrightarrow_{A_i} \pi_2
$$

Each instantiation of the universally quantified variable τ_j corresponds to one timepoint in which we want to check knowledge on. Therefore, we verify that k appears exactly once on the trace (first line of the formula). Then, we add to the knowledge set all traces that are equivalent (by the knowledge if

```
l ← 0
if h then
  o ← 1
else
  o ← o + 1
```

(a)

```
o ← 0
if h then
  o ← 1
else
  reg ← o + 1
  o ← reg
```

(b)

Fig. 2: Example Programs.

this agent, or by the common knowledge of all agents) until this timepoint. The formula outside the minimality condition verifies that for all traces in the set Y_j, the subformula φ_{j-1} holds thus enforcing the knowledge requirement on all traces in Y_j. In addition, it uses the property $\square\left(k_{\tau_j} \to \psi_j\right)$ to make sure that the temporal (non-knowledge) requirements hold at the same time for all traces in Y_j. Finally, we define $\varphi' = \forall\pi.\varphi_n$ where n is the number of knowledge operators in φ. Note that in general, the formula above can yield infinitely many sets of traces. In practical examples, e.g. the examples appear in this paper, we can write simplified formulas that reason about the specific problem at hand and only require a finite (usually 1) number of such sets. Also note that as the sets Y_j are unique, we do not need the quantification over least sets. $\square$

B.2 Additional Material To Section 4.2

Consider the system in Figure 2a (taken from [3]). The synchronous version of observational determinism (φ_{OD}) holds on this system: While we branch on the secret input h, the value of o is the same across all traces. In contrast, φ_{OD} does not hold on the system in Figure 2b as, in the second branch, the update occurs one step later. This, however, is not an accurate interpretation of φ_{OD} (assuming that an attacker only has access to the memory footprint and not the CPU registers or a timing channel), as any two traces are *stutter* equivalent (with respect to o). In AHLTL we can express an asynchronous version of OD as $\forall\pi_1.\forall\pi_2.\mathbf{E}\square(o_{\pi_1} \leftrightarrow o_{\pi_2})$ stating that all two traces can be aligned such they (globally) agree on o. This formula now holds on both Figure 2a and Figure 2b.

C Additional Material for Section 6

C.1 Muddy Children

We consider the following Hyper^{2}LTL$_{\text{fp}}$ formula which captures the common knowledge set after m steps.

$$\forall\pi \in \mathfrak{S}.$$
$$\Big(X, \curlyvee, \pi \triangleright X \land \forall\pi_1 \in X.\forall\pi_2 \in \mathfrak{S}.\big(\bigvee_{i\in[n]} \square^{\leq m}\, \pi_1 =_{\text{AP}_i} \pi_2\big) \Rightarrow \pi_2 \triangleright X\Big).$$
$$\forall\pi_1 \in X.\forall\pi_2 \in X. \bigwedge_{i\in[n]} i_{c_{\pi_1}} \leftrightarrow i_{c_{\pi_2}}$$

where we write $\square^{\leq m} \psi$ to assert that ψ holds in the first m steps. Here, AP_i are all propositions observable by child i, i.e., all variables expect the one that determines if i is muddy. Note that this formula falls within our fixpoint fragment of $\text{Hyper}^2\text{LTL}_{\text{fp}}$.

Further note that we express a hyperproperty on the knowledge set, i.e., compare pairs of traces in the knowledge set. This is not possible in logic's such as $\text{LTL}_{\mathsf{K},\mathsf{C}}$ in which we can only check if a trace property holds on the knowledge set.

C.2 Asynchronous Hyperproperties

We verify

$$\varphi_{OD} := \forall \pi_1. \forall \pi_2. \square(o_{\pi_1} \leftrightarrow o_{\pi_2}) \tag{4}$$

and the asynchronous version of it. In AHLTL [3] we can define this as follows:

$$\varphi_{OD} := \forall \pi_1. \forall \pi_2. \mathbf{E} \square(o_{\pi_1} \leftrightarrow o_{\pi_2})$$

In $\text{Hyper}^2\text{LTL}_{\text{fp}}$ we can express the above AHLTL formula as the following formula:

$$\begin{aligned} & \forall \pi_1 \in \mathfrak{S}. \forall \pi_2 \in \mathfrak{S}. \\ & \quad \big(X_1, \curlyvee, \pi_1 \triangleright X_1 \land \forall \pi \in X. \forall \pi \in \mathfrak{A}. ((o_\pi \leftrightarrow o_{\pi'}) \,\mathcal{U} \square(o_\pi \leftrightarrow \bigcirc o_{\pi'}) \rightarrow \pi' \triangleright X_1\big) \\ & \quad \big(X_2, \curlyvee, \pi_2 \triangleright X_2 \land \forall \pi \in X. \forall \pi' \in \mathfrak{A}. ((o_\pi \leftrightarrow o_{\pi'}) \,\mathcal{U} \square(o_\pi \leftrightarrow \bigcirc o_{\pi'}) \rightarrow \pi' \triangleright X_2\big) \\ & \qquad \exists \pi_1 \in X_1, \exists \pi_2 \in X_2. \square(o_{\pi_1} \leftrightarrow o_{\pi_2}) \end{aligned} \tag{5}$$

Note that this formula falls within our fixpoint fragment of $\text{Hyper}^2\text{LTL}_{\text{fp}}$.

In Table 1b, we check Equation (4) and Equation (5) on the two example programs from the introduction in [3] and the asynchronous program `Q1` from [5,9].

C.3 Mazurkiewicz Trace

Using our logic, we can also express many properties that reason about the class of (Mazurkiewicz) traces. The idea of trace is to abstract away from the concrete order of independent actions (letters). Let $I \subseteq \Sigma \times \Sigma$ be an independence relation on letters. That is, $(a, b) \in I$ iff interchanging the order of a and b has no effect (e.g., local actions for two concurrent processes). We say two traces t_1, t_2 are equivalent (written $t_1 \equiv_I t_2$) if we can rewrite t_1 into t_2 by flipping consecutive letters that are in I. For example if $(a, b) \in I$ then $xaby \equiv_I xbay$ for all $x \in \Sigma^*, y \in \Sigma^\omega$. The Mazurkiewicz trace of a concrete trace t is then defined as $[t]_I := \{t' \mid t \equiv_i t\}$.

Using Hyper^{2}LTL$_{\text{fp}}$ we can directly reason about the equivalence classes of $\equiv_I$. Consider the following (quantifier-free) formula $\varphi_I(\pi, \pi')$, stating that π and π' are identical up to one flip of consecutive independent actions.

$$(\pi =_{\text{AP}} \pi')\, \mathcal{W} \Big(\bigvee_{(x,y)\in I} x_\pi \wedge y_{\pi'} \wedge \bigcirc(y_\pi \wedge x_{\pi'}) \wedge \bigcirc\bigcirc\square(\pi =_{\text{AP}} \pi')\Big)$$

Here we write x_π for $x \in \Sigma = 2^{\text{AP}}$ for the formula $\bigwedge_{a\in X} a_\pi \wedge \bigwedge_{a\notin X} \neg a_\pi$. The formula asserts that both traces are equal, until one pair of independent actions is flipped, followed by, again, identical suffixes.

Using φ_I we can directly reason about Mazurkiewicz traces. Assume we are interested if for every (concrete) trace t that satisfies LTL property ϕ, all its equivalent traces satisfy ψ. We can express this in Hyper^{2}LTL$_{\text{fp}}$ as follows:

$$\begin{aligned}&\forall \pi \in \mathfrak{S}.(X, \curlyvee, \pi \triangleright X \wedge \forall \pi_1 \in X. \forall \pi_2 \in \mathfrak{A}. \varphi_I(\pi_1, \pi_2) \rightarrow \pi_2 \triangleright X).\\ &\qquad \forall \pi' \in X. \phi(\pi) \rightarrow \psi(\pi')\end{aligned} \tag{6}$$

That is, for all traces π, we compute the set X which contains all equivalent traces. This set should contain π, must be closed under φ_I, and is minimal w.r.t. those two properties. Note that this formula falls within our fixpoint fragment of Hyper^{2}LTL$_{\text{fp}}$.

Preliminary Experiments. The above formula is applicable to arbitrary independec relation, so our tool can be used to automatically check arbitrary properties on Mazurkiewicz traces. In our preliminary experiments, we focus on very simple Mazurkiewicz trace. We model a situation where we start with $\{a\}^1\{\}^\omega$ and can swap letters $\{a\}$ and $\{\}$. We acknowledge that the example is very simple, but nevertheless emphasize the complex trace-based reasoning possible with `HySO`. We then ask the following problems: We ask if, from every trace, a holds at most once on all traces in X. If we apply only iteration, `HySO` will move the unique a one step to the right in each iteration, i.e., after n steps the current under-approximation contains traces $\{a\}\{\}^\omega$, $\{\}^1\{a\}\{\}^\omega$, $\{\}^2\{a\}\{\}^\omega$, $\dots$, $\{\}^n\{a\}\{\}^\omega$. This fixpoint will never converge, so `HySO` would iterate forever. Instead, if we also enable learning of overapproximations, `HySO` automatically learns an invariant that proves the above property (which is instance SwapA in Table 2b). We can relax this requirement to only consider a fixed, finite prefix. SwapA$_n$ states that a *can* hold in any position in the first n steps (by using existential quantification over X). As expected, `HySO` can prove this property by iteration n times. Lastly, the violation SwapAViolation$_n$ states that any trace satisfies a within the first n steps. This obviously does not hold, but `HySO` requires n iterations to find a counterexample, i.e., a trace where a does not hold within the first n steps.