Model Checking Algorithms for Hyperproperties (invited paper)

Bernd Finkbeiner[0000−0002−4280−8441]

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
finkbeiner@cispa.saarland

Abstract. Hyperproperties generalize trace properties by expressing relations between multiple computations. Hyperpropertes include policies from information-flow security, like observational determinism or noninterference, and many other system properties including promptness and knowledge. In this paper, we give an overview on the model checking problem for temporal hyperlogics. Our starting point is the model checking algorithm for HyperLTL, a reduction to Büchi automata emptiness. This basic construction can be extended with propositional quantification, resulting in an algorithm for HyperQPTL. It can also be extended with branching time, resulting in an algorithm for HyperCTL*. However, it is not possible to have both extensions at the same time: the model checking problem of HyperQCTL* is undecidable. An attractive compromise is offered by MPL[E], i.e., monadic path logic extended with the equal-level predicate. The expressiveness of MPL[E] falls strictly between that of HyperCTL* and HyperQCTL*. MPL[E] subsumes both HyperCTL* and HyperKCTL*, the extension of HyperCTL* with the knowledge operator. We show that the model checking problem for MPL[E] is still decidable.

1 Introduction

In recent years, the linear-time and branching-time temporal logics have been extended to allow for the specification of hyperproperties [3,11,8,5,7]. Hyperproperties are a generalization of trace properties. Instead of properties of individual computations, hyperproperties express *relations* between multiple computations [4]. This makes it possible to reason uniformly about system properties like information flow, promptness, and knowledge.

In model checking, hyperproperties have played a significant role even before these new logics became available. An early insight was that the verification of a given system against properties that refer to multiple traces can be reduced to the verification of a *modified* system against properties over individual traces. The idea is to self-compose the given system a sufficient number of times. The resulting traces contain in each position a tuple of observations, each resulting from a different computation of the system. With this principle, certain hyperproperties like observational determinism and noninterference can be verified using model checking algorithms for standard linear and branching-time logics [13,1,18].

The development of new logics specifically for hyperproperties considerably broadened the range of hyperproperties that can be checked automatically. HyperLTL is an extension of linear-time temporal logic (LTL) with quantifiers over trace variables, which allow the formula to refer to multiple traces at the same time. For example, *noninterference* [12] between a secret input h and a public output o can be specified in HyperLTL by requiring that all pairs of traces π and π' that have, in every step, the same inputs except for h (i.e., all inputs in $I \setminus \{h\}$ are equal on π and π') also have the same output o at all times:

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

By combining universal and existential quantification, HyperLTL can also express properties like *generalized noninterference* (GNI) [15], which requires that for every pair of traces π and π', there is a third trace π'' that agrees with π on h and with π' on o:

$$\forall \pi . \forall \pi' . \exists \pi'' .\ \mathbf{G}\, (h_\pi = h_{\pi''}) \ \wedge\ \mathbf{G}\, (o_{\pi'} = o_{\pi''})$$

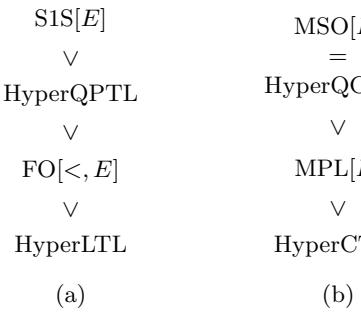
HyperLTL is the starting point of an entire hierarchy of *hyperlogics*, depicted in Fig. 1 and analyzed in detail in [5]. The hyperlogics are obtained from their classic counterparts with two principal extensions. The temporal logics LTL, QPTL, and CTL* are extended with quantifiers and variables over traces or paths, such that the formula can refer to multiple traces or paths at the same time; the first-order and second-order logics FO, S1S, MPL, and MSO are extended with the equal-level predicate E, which indicates that two points happen at the same time (albeit possibly on different computations of the system).

A key limitation of HyperLTL, as first pointed out by Bozzelli et al. [2], is that it is not possible to express promptness requirements, which say that there should exist a common deadline over all traces by which a certain eventuality is satisfied. Such properties can be expressed in FO$[<, E]$, monadic first-order logic of order extended with the equal-level predicate. FO$[<, E]$ is subsumed by the temporal logic HyperQPTL, which extends HyperLTL with quantification over propositions. The following HyperQPTL formula specifies the existence of a common deadline over all traces by which a certain predicate p must become true on all traces. The quantification over the proposition d, which expresses the common deadline, introduces a valuation of d that is *independent* of the choice of trace π:

$$\exists d . \forall \pi .\ \neg d\ \mathcal{U}\ (p_\pi \wedge \mathbf{F}\, d)$$

HyperQPTL captures the *ω-regular hyperproperties* [9]. Even more expressive is S1S$[E]$, monadic second order logic with one successor equipped with the equal-level predicate. While the model checking problem of HyperQPTL is still decidable, it becomes undecidable for S1S$[E]$. This is different from the case of trace properties, where S1S is equally expressive to QPTL, and both have decidable model checking problems.

Extending HyperLTL to branching time leads to the temporal logic HyperCTL* [3], which has the same syntax as HyperLTL, except that the quantifiers



(a) (b)

Fig. 1: The hierarchy of hyperlogics [5]: (a) linear time, (b) branching time.

refer to paths, rather than traces, and that path quantifiers may occur in the scope of temporal modalities. HyperCTL* is subsumed by monadic path logic equipped with the equal-level predicate (MPL[E]), which is a second-order logic where second-order quantifiers are restricted to full computation paths. MPL[E] in turn is contained in HyperQCTL*, the extension of HyperCTL* with propositional quantification. HyperQCTL* is as expressive as full monadic second-order logic with the equal-level predicate (MSO[E]) [5].

In this paper, we study this hierarchy of logics from the perspective of the model checking problem. Our starting point is the model checking algorithm for HyperLTL, which reduces the model checking problem to the language emptiness problem of a Büchi automaton [10]. The construction is similar to the idea of self-composition in that for every trace variable a separate copy of the system is introduced. Quantifiers are then eliminated by existential and universal projection on the language of the automaton. This basic construction can be extended with propositional quantification, which is also handled by projection. The construction can also be extended to branching time, by tracking the precise state of each computation, rather than just the trace label. However, it is not possible to implement both extensions at the same time: the model checking problem of HyperQCTL* is undecidable [5].

The undecidability of HyperQCTL* is unfortunate, because many interesting properties, such as branching-time knowledge, can be expressed in HyperQCTL*, but not in HyperCTL*. It turns out, however, that MPL[E], whose expressiveness lies strictly between HyperCTL* and HyperQCTL*, still has a decidable model checking problem. As the only original contribution of this paper (everything else is based on previously published results), we present the first model checking algorithm for MPL[E]. MPL[E] is a very attractive compromise. MPL[E] subsumes both HyperCTL* and HyperKCTL* [5], the extension of HyperCTL* with the knowledge operator.

2 HyperLTL

HyperLTL is a generalization of linear-time temporal logic (LTL). We quickly review the syntax and semantics of LTL and then describe the extension to HyperLTL. Let AP be a finite set of atomic propositions. A trace over AP is a map $t\colon \mathbb{N} \to 2^{\mathrm{AP}}$, denoted by $t(0)t(1)t(2)\cdots$. Let $(2^{\mathrm{AP}})^\omega$ denote the set of all traces over AP.

LTL. The formulas of linear-time temporal logic (LTL) [16] are generated by the following grammar:

$$\varphi \quad ::= \quad a \;\mid\; \neg\varphi \;\mid\; \varphi \wedge \varphi \;\mid\; \mathbf{X}\,\varphi \;\mid\; \varphi\,\mathbf{U}\,\varphi$$

where $a \in \mathrm{AP}$ is an *atomic proposition*, the Boolean connectives $\neg$ and $\wedge$ have the usual meaning, $\mathbf{X}$ is the temporal *next* operator, and $\mathbf{U}$ is the temporal *until* operator. We also consider the usual derived Boolean connectives, such as $\vee$, $\rightarrow$, and $\leftrightarrow$, and the derived temporal operators *eventually* $\mathbf{F}\,\varphi \equiv \mathbf{tt}\,\mathbf{U}\,\varphi$, *globally* $\mathbf{G}\,\varphi \equiv \neg\mathbf{F}\,\neg\varphi$, and *weak until*: $\varphi\,\mathcal{W}\,\psi \equiv (\varphi\,\mathbf{U}\,\psi) \vee \mathbf{G}\,\varphi$. The satisfaction of an LTL formula φ over a trace t at a position $i \in \mathbb{N}$, denoted by $t, i \models \varphi$, is defined as follows:

$$\begin{array}{lll}
t,i \models a & \text{iff} & a \in t(i),\\
t,i \models \neg\varphi & \text{iff} & t,i \not\models \varphi,\\
t,i \models \varphi_1 \wedge \varphi_2 & \text{iff} & t,i \models \varphi_1 \text{ and } t,i \models \varphi_2,\\
t,i \models \mathbf{X}\,\varphi & \text{iff} & t,i+1 \models \varphi,\\
t,i \models \varphi_1 \mathbf{U} \varphi_2 & \text{iff} & \exists k \geq i:\ t,k \models \varphi_2 \ \wedge\ \forall i \leq j < k:\ t,j \models \varphi_1.
\end{array}$$

We say that a trace t satisfies a sentence φ, denoted by $t \models \varphi$, if $t, 0 \models \varphi$. For example, the LTL formula $\mathbf{G}\,(a \rightarrow \mathbf{F}\,b)$ specifies that every position in which a is true must eventually be followed by a position where b is true.

HyperLTL. The formulas of HyperLTL [3] are generated by the grammar

$$\begin{aligned}
\varphi &::= \exists \pi.\ \varphi \;\mid\; \forall \pi.\ \varphi \;\mid\; \psi\\
\psi &::= a_\pi \;\mid\; \neg\psi \;\mid\; \psi \wedge \psi \;\mid\; \mathbf{X}\,\psi \;\mid\; \psi\,\mathbf{U}\,\psi
\end{aligned}$$

where a is an atomic proposition from a set AP and π is a trace variable from a set $\mathcal{V}$. Further Boolean connectives and the temporal operators $\mathbf{F}$, $\mathbf{G}$, and $\mathcal{W}$ are derived as for LTL. A sentence is a closed formula, i.e., the formula has no free trace variables.

The semantics of HyperLTL is defined with respect to a trace assignment, a partial mapping $\Pi\colon \mathcal{V} \to (2^{\mathrm{AP}})^\omega$. The assignment with empty domain is denoted by $\Pi_\emptyset$. Given a trace assignment Π, a trace variable π, and a trace t, we denote by $\Pi[\pi \to t]$ the assignment that coincides with Π everywhere but at π, which is mapped to t. The satisfaction of a HyperLTL formula φ over a trace assignment Π and a set of traces T at a position $i \in \mathbb{N}$, denoted by $T, \Pi, i \models \varphi$, is defined as follows:

$$
\begin{array}{lll}
T, \Pi, i \models a_\pi & \text{iff} & a \in \Pi(\pi)(i), \\
T, \Pi, i \models \neg\psi & \text{iff} & T, \Pi, i \not\models \psi, \\
T, \Pi, i \models \psi_1 \wedge \psi_2 & \text{iff} & T, \Pi, i \models \psi_1 \text{ and } T, \Pi, i \models \psi_2, \\
T, \Pi, i \models \mathbf{X}\,\psi & \text{iff} & T, \Pi, i+1 \models \psi, \\
T, \Pi, i \models \psi_1 \,\mathbf{U}\, \psi_2 & \text{iff} & \exists k \geq i : T, \Pi, k \models \psi_2 \\
 & & \wedge \forall i \leq j < k : T, \Pi, j \models \psi_1, \\
T, \Pi, i \models \exists \pi.\ \varphi & \text{iff} & \exists t \in T : T, \Pi[\pi \to t], i \models \psi, \\
T, \Pi, i \models \forall \pi.\ \varphi & \text{iff} & \forall t \in T : T, \Pi[\pi \to t], i \models \psi.
\end{array}
$$

We say that a set T of traces satisfies a sentence φ, denoted by $T \models \varphi$, if $T, \Pi_\emptyset, 0 \models \varphi$.

System properties. A *Kripke structure* is a tuple $K = (S, s_0, \delta, \mathrm{AP}, L)$ consisting of a set of states S, an initial state s_0, a transition function $\delta : S \to 2^S$, a set of *atomic propositions* AP, and a *labeling function* $L : S^* \to 2^{\mathrm{AP}}$ that assigns a set of atomic propositions that are true after a given sequence of states has been traversed. We require that each state has a successor, that is $\delta(s) \neq \emptyset$, to ensure that every execution of a Kripke structure can always be continued to infinity. In a *finite* Kripke structure, S is a finite set. We furthermore assume that in a finite Kripke structure, L only depends on the last state, so that L can also be given as a function $S \to 2^{\mathrm{AP}}$.

A *path* of a Kripke structure is an infinite sequence $s_0 s_1 \ldots \in S^\omega$ such that s_0 is the initial state of K and $s_{i+1} \in \delta(s_i)$ for all $i \in \mathbb{N}$. By $\mathit{Paths}(K, s)$, we denote the set of all paths of K starting in state $s \in S$. A *trace* of a path $\sigma = s_0 s_1 \ldots$ is a sequence of labels $l_0 l_1 \ldots$ with $l_i = L(s_0 s_1 \ldots s_i)$ for all $i \in \mathbb{N}$. $\mathit{Tr}(K, s)$ is the set of all traces of paths of a Kripke structure K starting in state s. A Kripke structure K with initial state s_0 satisfies an LTL formula φ, denoted by $K \models \varphi$ iff for all traces $\pi \in \mathit{Tr}(K, s_0)$, it holds that $\pi \models \varphi$. Likewise, the Kripke structure satisfies a HyperLTL formula φ, also denoted by $K \models \varphi$, iff $\mathit{Tr}(K, s_0) \models \varphi$.

Model checking. The HyperLTL model checking problem is to decide, for a given finite Kripke structure K and a given HyperLTL formula ψ, whether or not $K \models \psi$. The following basic construction (described in more detail in [10]) reduces the model checking problem to the language emptiness problem of a Büchi automaton: the given Kripke structure satisfies the formula if and only if the language of the resulting automaton is empty.

The construction starts by negating ψ, so that it describes the existence of an error. Since we assume that a HyperLTL formula begins with a quantifier prefix, this means that we dualize the quantifiers and then negate the inner LTL formula. Let us assume that the resulting HyperLTL formula has the form $Q_n \pi_{n-1} Q_2 \pi_{n-1} \ldots Q_1 \pi_1.\ \varphi$ where $Q_1, Q_2, \ldots Q_n$ are trace quantifiers in $\{\exists, \forall\}$ and φ is a quantifier-free formula over atomic propositions indexed by trace variables $\{\pi_1, \ldots \pi_n\}$.

Similar to standard LTL model checking, we convert the LTL formula φ into an equivalent Büchi automaton $\mathcal{A}_0$ over the alphabet $(2^{\mathrm{AP}})^n$. Each letter

is a tuple of n sets of atomic propositions, where the ith element of the tuple represents the atomic propositions of trace π_i.

Next, the algorithm eliminates the quantifiers. For this purpose, it carries out n steps that each eliminate one component from the tuple of the input alphabet. In the ith step, we eliminate the ith component, corresponding to trace variable π_i. Let us consider the ith step. Over the previous steps, the automaton $\mathcal{A}_{i-1}$ over alphabet $(2^{\text{AP}})^{(n-i)}$ has been constructed, and now the first component of the tuple corresponds to π_i. If the trace quantifier Q_i is existential, we intersect $\mathcal{A}_{i-1}$ with the Kripke structure K so that, in the sequence of letters, the first component of the tuple is chosen consistently with some path in K. Subsequently, we eliminate the first component of the tuple by existential projection on the automaton. If Q_i is universal, then we combine $\mathcal{A}_{i-1}$ with the Kripke structure K so that only sequences in which the first component is chosen consistently with some path in K need to be accepted by $\mathcal{A}_{i-1}$. Subsequently, we eliminate the first component of the tuple by universal projection on the automaton. This results in the next automaton $\mathcal{A}_i$.

After n such steps, all quantifiers have been eliminated and the language of the resulting automaton is over the one-letter alphabet (consisting of the empty tuple). The HyperLTL formula is satisfied if and only if the language of automaton $\mathcal{A}_n$ is empty.

3 HyperQPTL

HyperQPTL [17,5] extends HyperLTL with quantification over atomic propositions. To easily distinguish quantification over traces $\exists\pi, \forall\pi$ and quantification over propositions $\exists p, \forall p$, we use boldface for the latter. The formulas of HyperQPTL are generated by the following grammar:

$$\varphi ::= \exists \pi.\ \varphi \ \mid\ \forall \pi.\ \varphi \ \mid\ \psi \ \mid\ \exists p.\ \varphi \ \mid\ \forall p.\ \varphi \ \mid\ \psi$$
$$\psi ::= a_\pi \ \mid\ p \ \mid\ \neg\psi \ \mid\ \psi \wedge \psi \ \mid\ \mathbf{X}\,\psi \ \mid\ \mathbf{F}\,\psi$$

where $a, p \in \text{AP}$ and $\pi \in \mathcal{V}$. The semantics of HyperQPTL corresponds to the semantics of HyperLTL with additional rules for propositional quantification:

$$\begin{aligned} T, \Pi, i &\models \exists q.\ \varphi \quad \text{iff} \quad \exists t \in (2^{\{q\}})^\omega.\ T, \Pi[\pi_q \mapsto t], i \models \varphi \\ T, \Pi, i &\models \forall q.\ \varphi \quad \text{iff} \quad \forall t \in (2^{\{q\}})^\omega.\ T, \Pi[\pi_q \mapsto t], i \models \varphi \\ T, \Pi, i &\models q \qquad\ \text{iff} \quad q \in \Pi(\pi_q)(i). \end{aligned}$$

Expressiveness. As discussed in the introduction, HyperQPTL can express *promptness* [14], which states that there is a bound, common for all traces, until which an eventuality has to be fulfilled. Another common type of property that can be expressed in HyperQPTL is *knowledge*. Epistemic temporal logics extend temporal logics with a so-called *knowledge* operator $\mathcal{K}_A\varphi$, denoting that an agent A knows φ. HyperQPTL can be extended to HyperQPTL$_K$ as follows [17]:

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

In this definition, $t[0,i]$ denotes the prefix of a trace t up to position i. Two sequences t, t' are equivalent with respect to agent A, denoted by $t = At'$, if A cannot distinguish t and t'. We assume that A is given as a set of atomic propositions $A \subseteq \text{AP}$. Then $t =_A t'$ holds if t and t' agree on all propositions in A.

As shown in [17], the knowledge operator can be eliminated, resulting in an equivalent HyperQPTL formula. The idea is to replace an application of the knowledge operator $\mathcal{K}_{A,\pi}\psi$ with an existentially quantified proposition u and add the following requirement to ensure that u is only true at positions where the knowledge formula is satisfied:

$$\forall r.\, \forall \pi'.\, ((r\ \mathcal{U}\ (u \wedge r \wedge \bigcirc\Box\neg r)) \wedge \Box(r \rightarrow A_\pi = A_{\pi'}) \rightarrow \Box(r \wedge \bigcirc\neg r \rightarrow \psi[\pi'/\pi]))$$

In this definition, $A_\pi = A_{\pi'}$ is an abbreviation for the conjunction over all propositions in A that ensures that each proposition has the same value in π and in π'. For each position where the knowledge formula is claimed to be true, the universally quantified proposition r changes from true to false at exactly that position, thus marking the prefix leading to this point. The knowledge formula is then true iff ψ holds on all traces π' that agree with respect to A on the prefix.

HyperQPTL is also strictly more expressive than FO$[<, E]$, the extension of the first-order logic of order with the equal-level predicate E [5]. Given a set V_1 of first-order variables, the formulas φ of FO$[<, E]$ are generated by the following grammar [11]:

$$\begin{aligned}\varphi &::= \psi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x.\varphi \\ \psi &::= P_a(x) \mid x < y \mid x = y \mid E(x,y),\end{aligned}$$

where $a \in AP$ and $x, y \in V_1$. We interpret FO$[<, E]$ formulas over a set of traces T. We assign first-order variables to elements from the domain $T \times \mathbb{N}$. We define the satisfaction relation $T, \mathcal{V}_1 \models \varphi$ with respect to a valuation $\mathcal{V}_1$ assigning all free variables in V_1' as follows:

$$\begin{aligned}
T, \mathcal{V}_1 &\models P_a(x) && \text{iff } a \in t(n) \text{ where } (t,n) = \mathcal{V}_1(x) \\
T, \mathcal{V}_1 &\models x < y && \text{iff } t_1 = t_2 \wedge n_1 < n_2 \text{ where } (t_1,n_1) = \mathcal{V}_1(x) \text{ and } (t_2,n_2) = \mathcal{V}_1(y) \\
T, \mathcal{V}_1 &\models x = y && \text{iff } \mathcal{V}_1(x) = \mathcal{V}_1(y) \\
T, \mathcal{V}_1 &\models E(x,y) && \text{iff } n_1 = n_2 \text{ where } (t_1,n_1) = \mathcal{V}_1(x) \text{ and } (t_2,n_2) = \mathcal{V}_1(y) \\
T, \mathcal{V}_1 &\models \neg\varphi && \text{iff } T, \mathcal{V}_1 \not\models \varphi \\
T, \mathcal{V}_1 &\models \varphi_1 \vee \varphi_2 && \text{iff } T, \mathcal{V}_1 \models \varphi_1 \text{ or } T, \mathcal{V}_1 \models \varphi_2 \\
T, \mathcal{V}_1 &\models \exists x.\varphi && \text{iff } \exists (t,n) \in T \times \mathbb{N}. \\
& && \quad T, \mathcal{V}_1[x \mapsto (t,n)] \models \varphi,
\end{aligned}$$

where $\mathcal{V}_1[x \mapsto v]$ updates a valuation. A trace set T satisfies a closed FO$[<, E]$ formula φ, written $T \models \varphi$, if $T, \emptyset \models \varphi$, where $\emptyset$ denotes the empty valuation.

Model checking. The only required modification to the model checking algorithm described in Section 2 is the treatment of the propositional quantifiers. Since the

valuation of the propositions is not restricted by the given Kripke structure, we omit the intersection with the Kripke structure for quantified propositions, and instead eliminate the quantifier by existential or universal projection only.

4 Beyond HyperQPTL

The model checking problems of linear-time hyperlogics beyond HyperQPTL quickly become undecidable. Two examples of such logics are HyperQPTL$^+$ and S1S[E].

HyperQPTL$^+$. HyperQPTL$^+$ [9] differs from HyperQPTL in the role of the propositional quantification. Rather than interpreting the quantified propositions with an additional sequence of values, HyperQPTL$^+$ modifies the interpretation on the existing traces. The syntax of HyperQPTL$^+$ is thus slightly simpler, because also the quantified propositions appear indexed with trace variables:

$$\begin{aligned}\varphi &::= \forall \pi.\, \varphi \;\mid\; \exists \pi.\, \varphi \;\mid\; \forall a.\, \varphi \;\mid\; \exists a.\, \varphi \;\mid\; \psi \\ \psi &::= a_\pi \;\mid\; \neg\psi \;\mid\; \psi \vee \psi \;\mid\; \mathbf{X}\,\psi \;\mid\; \mathbf{F}\,\psi \;.\end{aligned}$$

In the semantics, the rules for propositional quantification are changed accordingly:

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

S1S[E]. S1S[E] is monadic second-order logic with one successor (S1S) extended with the equal-level predicate. Let $V_1 = \{x_1, x_2, \ldots\}$ be a set of first-order variables, and $V_2 = \{X_1, X_2, \ldots\}$ a set of second-order variables. The formulas φ of S1S[E] are generated by the following grammar:

$$\begin{aligned}\tau &::= x \;\mid\; \mathit{min}(x) \;\mid\; \mathit{Succ}(\tau) \\ \varphi &::= \tau \in X \;\mid\; \tau = \tau \;\mid\; E(\tau,\tau) \;\mid\; \neg\varphi \;\mid\; \varphi \vee \varphi \;\mid\; \exists x.\varphi \;\mid\; \exists X.\varphi,\end{aligned}$$

where $x \in V_1$ is a first-order variable, *Succ* denotes the successor relation, and $\mathit{min}(x)$ indicates the minimal element of the traces addressed by x. Furthermore, $E(\tau,\tau)$ is the equal-level predicate and $X \in V_2 \cup \{X_a \mid a \in AP\}$. We interpret S1S[$E$] formulas over a set of traces T. As for FO[$<, E$], the domain of the first-order variables is $T \times \mathbb{N}$. Let $\mathcal{V}_1 : V_1 \rightarrow T \times \mathbb{N}$ and $\mathcal{V}_2 : V_2 \rightarrow 2^{(T \times \mathbb{N})}$ be the first-order and second-order valuation, respectively. The value of a term is defined as follows:

$$\begin{aligned}[x]_{\mathcal{V}_1} &= \mathcal{V}_1(x) \\ [\mathit{min}(x)]_{\mathcal{V}_1} &= (\mathit{proj}_1(\mathcal{V}_1(x)), 0) \\ [S(\tau)]_{\mathcal{V}_1} &= (\mathit{proj}_1([\tau]_{\mathcal{V}_1}), \mathit{proj}_2([\tau]_{\mathcal{V}_1}) + 1),\end{aligned}$$

where $proj_1$ and $proj_2$ denote the projection to the first and second component, respectively. Let φ be an S1S[E] formula with free first-order and second-order variables $V_1' \subseteq V_1$ and $V_2' \subseteq V_2 \cup \{X_a \mid a \in AP\}$, respectively. We define the satisfaction relation $T, \mathcal{V}_1, \mathcal{V}_2 \models \varphi$ with respect to two valuations $\mathcal{V}_1, \mathcal{V}_2$ assigning all free variables in V_1' and V_2' as follows:

$$
\begin{aligned}
&T, \mathcal{V}_1, \mathcal{V}_2 \models \tau \in X && \text{iff } [\tau]_{\mathcal{V}_1} \in \mathcal{V}_2(X)\\
&T, \mathcal{V}_1, \mathcal{V}_2 \models \tau_1 = \tau_2 && \text{iff } [\tau_1]_{\mathcal{V}_1} = [\tau_2]_{\mathcal{V}_1}\\
&T, \mathcal{V}_1, \mathcal{V}_2 \models E(\tau_1, \tau_2) && \text{iff } proj_2([\tau_1]_{\mathcal{V}_1}) = proj_2([\tau_2]_{\mathcal{V}_1})\\
&T, \mathcal{V}_1, \mathcal{V}_2 \models \neg\varphi && \text{iff } T, \mathcal{V}_1, \mathcal{V}_2 \not\models \varphi\\
&T, \mathcal{V}_1, \mathcal{V}_2 \models \varphi_1 \vee \varphi_2 && \text{iff } T, \mathcal{V}_1, \mathcal{V}_2 \models \varphi_1 \text{ or } T, \mathcal{V}_1, \mathcal{V}_2 \models \varphi_2\\
&T, \mathcal{V}_1, \mathcal{V}_2 \models \exists x.\varphi && \text{iff } \exists (t,n) \in T \times \mathbb{N}.\\
&&& \quad T, \mathcal{V}_1[x \mapsto (t,n)], \mathcal{V}_2 \models \varphi\\
&T, \mathcal{V}_1, \mathcal{V}_2 \models \exists X.\varphi && \text{iff } \exists A \subseteq T \times \mathbb{N}.\\
&&& \quad T, \mathcal{V}_1, \mathcal{V}_2[X \mapsto A] \models \varphi,
\end{aligned}
$$

where $\mathcal{V}_i[x \mapsto v]$ updates a valuation. A trace set T satisfies a closed S1S[E] formula φ, written $T \models \varphi$, if $T, \emptyset, \mathcal{V}_2 \models \varphi$, where $\emptyset$ denotes the empty first-order valuation and $\mathcal{V}_2$ assigns each free X_a in φ to the set $\{(t,n) \in T \times \mathbb{N} \mid a \in t[n]\}$.

Model checking. The model checking problems of HyperQPTL$^+$ and S1S[E] are both undecidable, as shown in [9] and [5], respectively.

5 HyperCTL*

Extending the path quantifiers of CTL* by *path variables* leads to the logic HyperCTL*, which subsumes both HyperLTL and CTL*. The formulas of HyperCTL* are generated by the following grammar:

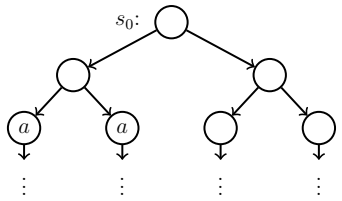
$$\varphi \quad ::= \quad a_\pi \quad | \quad \neg\varphi \quad | \quad \varphi \vee \varphi \quad | \quad \bigcirc \varphi \quad | \quad \varphi \,\mathbf{U}\, \varphi \quad | \quad \exists \pi.\, \varphi$$

We require that temporal operators only occur inside the scope of path quantifiers. The semantics of HyperCTL* is given in terms of assignments of variables to *paths*, which are defined analogously to trace assignments. Given a Kripke structure K, the satisfaction of a HyperCTL* formula φ at a position $i \in \mathbb{N}$, denoted by $K, \Pi, i \models \varphi$, is defined as follows:

$$
\begin{aligned}
&K, \Pi, i \models a_\pi && \text{iff} && a \in L\big(\Pi(\pi)[0 \dots i]\big),\\
&K, \Pi, i \models \neg\varphi && \text{iff} && \Pi, K, i \not\models \varphi,\\
&K, \Pi, i \models \varphi_1 \vee \varphi_2 && \text{iff} && K, \Pi, i \models \varphi_1 \text{ or } K, \Pi, i \models \varphi_2,\\
&K, \Pi, i \models \bigcirc \varphi && \text{iff} && K, \Pi, i+1 \models \varphi,\\
&K, \Pi, i \models \varphi_1 \,\mathbf{U}\, \varphi_2 && \text{iff} && \exists k \geq i:\ K, \Pi, k \models \varphi_2 \text{ and}\\
&&&&& \forall i \leq j < k:\ K, \Pi, j \models \varphi_1,\\
&K, \Pi, i \models \exists \pi.\, \varphi && \text{iff} && \exists p \in Paths(K, \Pi(\varepsilon)(i)):\\
&&&&& K, \Pi[\pi \mapsto p,\ \varepsilon \mapsto p], i \models \varphi,
\end{aligned}
$$

where ε is a special path variable that denotes the path most recently added to Π (i.e., closest in scope to π). For the empty assignment $\Pi_\emptyset$, we define $\Pi_\emptyset(\varepsilon)(i)$ to yield the initial state. A Kripke structure $K = (S, s_0, \delta, \mathrm{AP}, L)$ satisfies a HyperCTL* formula φ, denoted with $K \models \varphi$, iff $K, \Pi_\emptyset \models \varphi$.

Expressiveness. HyperCTL* can express the flow of information that appears in different branches of the computation tree. Consider, for example, the following Kripke structure (taken from [8]):



An observer who sees a can infer which branch was taken in the first nondeterministic choice, but not which branch was taken in the second nondeterministic choice. This is expressed by the HyperCTL* formula

$$\forall \pi.\, \mathbf{X}\, \forall \pi'.\, \mathbf{X}\, (a_\pi \leftrightarrow a_{\pi'}).$$

Model checking. The modification to the model checking algorithm from Section 2 needed to take care of branching time is to change to alphabet of the automata from $(2^{\mathrm{AP}})^n$, i.e., tuples of sets of atomic propositions, to S^n, i.e., tuples of states of the Kripke structure. The model checking algorithm is described in detail in [10]. The algorithm again starts by translating the inner LTL formula φ of the negated specification into an equivalent Büchi automaton $\mathcal{A}_0$ over the alphabet $(2^{\mathrm{AP}})^n$; this automaton is then translated into an automaton over alphabet S^n by applying the labeling function L to the individual positions of the tuple. The algorithm then proceeds as described in Section 2, eliminating in each step one path quantifier. In the elimination of the quantifier, the automaton is combined as before with the Kripke structure, ensuring that the state sequence corresponds to a path in the Kripke structure. After n steps, all quantifiers have been eliminated, and the language of the resulting automaton is, as before. over the one-letter alphabet (consisting of the empty tuple). The HyperCTL* formula is satisfied if and only if the language of the resulting automaton is empty.

6 HyperQCTL*

HyperQCTL* [5] extends HyperCTL* with quantification over atomic propositions. The formulas of HyperQCTL* are generated by the following grammar:

$$\varphi \quad ::= \quad a_\pi \quad | \quad \neg\varphi \quad | \quad \varphi \vee \varphi \quad | \quad \bigcirc \varphi \quad | \quad \varphi\, \mathbf{U}\, \varphi \quad | \quad \exists \pi.\, \varphi \quad | \quad \exists p.\, \varphi$$

where $a, p \in \mathrm{AP}$ and $\pi \in \mathcal{V}$. The semantics of HyperQCTL* corresponds to the semantics of HyperCTL* with an additional rule for propositional quantification.

In QPTL, a propositional quantifier over a proposition p determines a sequence in $(2^p)^\omega$; i.e., the value of the proposition depends on the position in the sequence. In HyperQCTL*, the quantification modifies the interpretation on the entire computation tree.

$$K, \Pi, i \models \exists q.\varphi \text{ iff } \exists L' : S^* \to 2^{AP \cup \{q\}}.\ \forall w \in S^*.\\ L'(w) =_{AP \setminus \{q\}} L(w) \wedge K[L'/L], \Pi, i \models \varphi.$$

We say that a Kripke structure K satisfies a HyperQCTL* formula φ, written $K \models \varphi$, if $K, \emptyset, 0 \models \varphi$.

Expressiveness. HyperQCTL* is strictly more expressive than HyperCTL*. In particular, HyperQCTL* subsumes the extension of HyperCTL* with the knowledge operator. The formula $K_{A,\pi}\varphi$ states that the agent who can observe the propositions $A \subseteq$ AP on path π knows that φ holds. The semantics of $K_{A,\pi}$ is defined (analogously to the linear-time version in Section 3) as follows:

$$K, \Pi, i \models \mathcal{K}_{A,\pi}\varphi \quad \text{iff} \quad \forall p \in \mathit{Paths}(K, s_0).\ \Pi(\pi)[0,i] =_A p[0,i] \to \\ T, \Pi[\pi \mapsto p], i \models_K \varphi$$

HyperQCTL* also has the same expressiveness as second-order modadic logic equipped with the equal-level predicate (MSO[E]), i.e., the extension of FO[$<, E$] (as defined in Section 3) with second-order quantification [5].

Model checking. The model checking problem of HyperQCTL* is undecidable [5].

7 Monadic Path Logic

Monadic path logic equipped with the equal-level predicate (MPL[E]) is the extension of FO[$<, E$] (as defined in Section 3) with second-order quantification, where the second-order quantification is restricted to full paths in the Kripke structure.

Let $V_1 = \{x_1, x_2, \ldots\}$ be a set of first-order variables, and $V_2 = \{X_1, X_2, \ldots\}$ a set of second-order variables. The formulas φ of MPL[E] are generated by the following grammar:

$$\varphi ::= \psi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x.\varphi \mid \exists X.\varphi \\ \psi ::= P_a(x) \mid x < y \mid x = y \mid x \in X \mid E(x, y),$$

where $a \in AP$, $x, y \in V_1$, and $X \in V_2$. In the semantics of MPL[E], we assign first-order variables to sequences of states that form a prefix of a path in the Kripke structure, and second-order variables to the infinite prefix-closed sets of prefixes of the paths of the Kripke structure.

We define the satisfaction relation $K, \mathcal{V}_1, \mathcal{V}_2 \models \varphi$ for a Kripke structure K and two valuations $\mathcal{V}_1, \mathcal{V}_2$ as follows:

$$
\begin{array}{lll}
K, \mathcal{V}_1, \mathcal{V}_2 \models P_a(x) & \text{iff} & a \in L(\mathcal{V}_1(x)) \\
K, \mathcal{V}_1, \mathcal{V}_2 \models x < y & \text{iff} & \mathcal{V}_1(x) \sqsubseteq \mathcal{V}_1(y) \\
K, \mathcal{V}_1, \mathcal{V}_2 \models x = y & \text{iff} & \mathcal{V}_1(x) = \mathcal{V}_1(y) \\
K, \mathcal{V}_1, \mathcal{V}_2 \models x \in X & \text{iff} & \mathcal{V}_1(x) \in \mathcal{V}_2(X) \\
K, \mathcal{V}_1, \mathcal{V}_2 \models E(x, y) & \text{iff} & |\mathcal{V}_1(x)| = |\mathcal{V}_1(y)| \\
K, \mathcal{V}_1, \mathcal{V}_2 \models \neg\varphi & \text{iff} & K, \mathcal{V}_1, \mathcal{V}_2 \not\models \varphi \\
K, \mathcal{V}_1, \mathcal{V}_2 \models \varphi_1 \vee \varphi_2 & \text{iff} & K, \mathcal{V}_1, \mathcal{V}_2 \models \varphi_1 \vee K, \mathcal{V}_1, \mathcal{V}_2 \models \varphi_2 \\
K, \mathcal{V}_1, \mathcal{V}_2 \models \exists x.\varphi & \text{iff} & \exists p \in S^*, p' \in \mathit{Paths}(K, s_0).\ p \sqsubseteq p' \wedge \\
 & & K, \mathcal{V}_1[x \mapsto p], \mathcal{V}_2 \models \varphi \\
K, \mathcal{V}_1, \mathcal{V}_2 \models \exists X.\varphi & \text{iff} & \exists p \in \mathit{Paths}(K, s_0).\ \mathcal{V}_1, \mathcal{V}_2[X \mapsto \mathit{Prefixes}(p)] \models \varphi
\end{array}
$$

where $\mathcal{V}_i[x \mapsto v]$ updates a valuation, $p_1 \sqsubseteq p_2$ denotes that p_1 is a prefix of p_2, and $\mathit{Prefixes}(p)$ is the set of prefixes of p. A Kripke structure K satisfies a closed MPL[E] formula φ, written $K \models \varphi$, if $T, \emptyset, \mathcal{V}_2 \models \varphi$, where $\emptyset$ denotes the empty first-order valuation and $\mathcal{V}_2$ assigns each free X_a in φ to the set $\{p \in S^* \mid a \in L(p)\}$.

Expressiveness. The expressiveness of MPL[E] falls strictly between HyperCTL* and HyperQCTL*. Like HyperQCTL*, MPL[E] can, however, express the properties of HyperKCTL*, i.e., the extension of HyperCTL* with the knowledge operator [5].

Model checking. Similar to the model checking algorithm of Section 2, we reduce the model checking problem of MPL[E] to the language emptiness problem of a Büchi automaton. Let φ be the negation of the given formula. We translate φ into an automaton $\mathcal{A}$ over the tuple alphabet $(S \cup \{\bot\})^{\mathcal{V}_1 \cup \mathcal{V}_2}$ such that the language of $\mathcal{A}$ is empty iff the original formula is satisfied by the Kripke structure. The automaton is constructed recursively as follows:

- If $\varphi = P_a(x)$, then $\mathcal{A}$ accepts all infinite sequences where the first time the component of component of x becomes $\bot$ at some point, and stays $\bot$ from thereon after, and a is contained in $L(w)$ where w is the sequence of states in x's component up to that point.
- If $\varphi = x < y$, then $\mathcal{A}$ accepts all infinite sequences where the components of x and y each become and stay $\bot$ at some point, and until x becomes $\bot$ the components are the same.
- If $\varphi = x{=}y$, then $\mathcal{A}$ accepts all infinite sequences where the components of x and y each become and stay $\bot$ at the same point, and until then the components are the same.
- If $\varphi = x \in X$, then $\mathcal{A}$ accepts all infinite sequences where the component of x becomes and stays $\bot$ at some point, and until then the components of x and X are the same.

- If $\varphi = E(x, y)$, then $\mathcal{A}$ accepts all infinite sequences where the components of x and y become and stay $\bot$ at the same point.
- If $\varphi = \neg\psi$, then we first compute and negate the automaton for ψ. $\mathcal{A}$ is then the intersection of that automaton with an automaton that ensures that, for every $x \in \mathcal{V}_1$, the component of x eventually becomes and stays $\bot$.
- If $\varphi = \exists x.\psi$, then we first compute the automaton for ψ. We then combine the automaton with the Kripke structure to ensure that the component for x forms a prefix of a path in the Kripke structure and ends in $\bot$. $\mathcal{A}$ is then the existential projection of that automaton, where the component for x is eliminated.
- If $\varphi = \exists X.\psi$, then we also first compute the automaton for ψ. We then combine the automaton with the Kripke structure to ensure that the component for X forms a full path in the Kripke structure. $\mathcal{A}$ is then the existential projection of that automaton, where the component for X is eliminated.

8 Conclusions

We have studied the hierarchy of hyperlogics from the perspective of the model checking problem. For the logics considered here, HyperQPTL is clearly the most interesting linear-time logic, because it can still be checked using the basic model checking algorithm, while for more expressive logics like HyperQPTL$^+$ and S1S[E] the model checking problem is already undecidable. Among the branching-time logics, MPL[E] has a similar position, more expressive than HyperCTL*, but, unlike HyperQCTL*, still with a decidable model checking problem.

From a practical point of view, the key challenge that needs to be addressed in all these logics is the treatment of quantifier alternations. In the model checking algorithm quantifier alternations lead to alternations between existential and universal projection on the constructed automaton. Such alternations can in theory be implemented using complementation; in practice, however, the exponential cost of complementation is too expensive. Model checking implementations like MCHyper therefore instead rely on quantifier elimination via strategies [6]. In this approach, the satisfaction of a formula of the form $\varphi = \forall\pi_1.\exists\pi_2.\psi$ is analyzed as a game between a *universal* player, who chooses π_1, and an *existential* player, who chooses π_2. The formula φ is satisfied if the existential player has a strategy that ensures that ψ becomes true.

Acknowledgements

Most of the work reported in this paper has previously appeared in various publications [3,5,6,9,10,11]. I am indebted to my coauthors Michael R. Clarkson, Norine Coenen, Christopher Hahn, Jana Hofmann, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, César Sánchez, Leander Tentrup, and Martin Zimmermann. This work was partially supported by the Collaborative Research Center "Foundations of Perspicuous Software Systems" (TRR: 248, 389792660) and the European Research Council (ERC) Grant OSARES (No. 683300).

References

1. Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. *Math. Struct. Comput. Sci.*, 21(6):1207–1252, 2011.
2. Laura Bozzelli, Bastien Maubert, and Sophie Pinchinat. Unifying hyper and epistemic temporal logics. In Andrew M. Pitts, editor, *FoSSaCS 2015*, volume 9034 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2015.
3. Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *POST 2014*, pages 265–284, 2014.
4. Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010.
5. Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. The hierarchy of hyperlogics. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019.
6. Norine Coenen, Bernd Finkbeiner, César Sánchez, and Leander Tentrup. Verifying hyperliveness. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 121–139. Springer, 2019.
7. Rayna Dimitrova, Bernd Finkbeiner, and Hazem Torfah. Probabilistic hyperproperties of markov decision processes. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis*, pages 484–500, Cham, 2020. Springer International Publishing.
8. Bernd Finkbeiner. Temporal hyperproperties. *Bulletin of the EATCS*, 123, 2017.
9. Bernd Finkbeiner, Christopher Hahn, Jana Hofmann, and Leander Tentrup. Realizing ω-regular hyperproperties. In *Computer Aided Verification*, pages 40–63, Cham, 2020. Springer International Publishing.
10. Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In Daniel Kroening and Corina S. Pasareanu, editors, *CAV 2015 (Part I)*, volume 9206 of *Lecture Notes in Computer Science*, pages 30–48. Springer, 2015.
11. Bernd Finkbeiner and Martin Zimmermann. The first-order logic of hyperproperties. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 30:1–30:14, 2017.
12. J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, April 1982.
13. M. Huisman, P. Worah, and K. Sunesen. A temporal logic characterisation of observational determinism. In *Proc. IEEE Computer Security Foundations Workshop*, pages 3–15, July 2006.
14. Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. From liveness to promptness. *Formal Methods in System Design*, 34(2):83–103, 2009.
15. Daryl McCullough. Noninterference and the composability of security properties. In *Proc. IEEE Symposium on Security and Privacy*, pages 177–186, April 1988.
16. Amir Pnueli. The Temporal Logic of Programs. In *FOCS 1977*, pages 46–57, 1977.
17. Markus N. Rabe. *A Temporal Logic Approach to Information-flow Control*. PhD thesis, Saarland University, 2016.
18. Ron van der Meyden and Chenyi Zhang. Algorithmic verification of noninterference properties. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 168:61–75, February 2007.