Model Checking Timed Hyperproperties in Discrete-time Systems*

Borzoo Bonakdarpour[1], Pavithra Prabhakar[2], and César Sánchez[3]

[1] Iowa State University, USA, borzoo@iastate.edu
[2] Kansas State University, USA, pprabhakar@ksu.edu
[3] IMDEA Software Institute, Spain, cesar.sanchez@imdea.org

Abstract. Many important timed requirements of computing systems cannot be described by the behavior of individual execution traces. Examples include countermeasures to deal with side-channel timing attacks and service-level agreements, which are examples of *timed hyperproperties.* In this paper, we propose the temporal logic HyperMTL, that extends MTL by allowing explicit and simultaneous quantification over multiple timed traces in the point-wise semantics. We demonstrate the application of HyperMTL in expressing important properties in information-flow security and cyber-physical systems. We also introduce a model checking algorithm for a nontrivial fragment of HyperMTL by reducing the problem to model checking untimed hyperproperties.

1 Introduction

There has been tremendous progress in automated reasoning about *trace properties* in the past three decades. These properties were classified by Alpern and Schneider [3] into *safety* and *liveness* properties. Temporal logics like LTL [31] and CTL [11] were crafted to give formal syntax and semantics of trace properties, and many verification algorithms and tools were developed to reason about these logics (see [11,34,9,10,12,29,5,4]). However, many interesting requirements are not trace properties. For example, information-flow security policies such as *noninterference* [25] and *observational determinism* [35] cannot be expressed as properties of individual execution traces. Also, service level agreement requirements (e.g., mean response time and percentage uptime) that use statistics across all executions of a system are not trace properties. Rather, they are properties of sets of execution traces. These requirements are *hyperproperties* [14]. Temporal logics, like HyperLTL and HyperCTL* [13], and probabilistic variants, like HyperPCTL [1], have been proposed to reason about temporal hyperproperties.

Hyperproperties can also be timed, i.e., they explicitly stipulate the timing relation of independent executions. An example of a timed hyperproperty is countermeasures against *timing channels.* A timing channel is one through which an attacker learns sensitive information by observing the time at which

* This research has been partially supported by the United States NSF SaTC Award 1813388, NSF CAREER Award 1552668, ONR YIP Award N000141712577, and by the Madrid Regional Government under project "S2018/TCS-4339 (BLOQUES-CM)" by Spanish National Project "BOSCO (PGC2018-102210-B-100)".

publicly observable events occur. A *timing leak* is an information leak through a timing channel. For instance, consider the program in Fig. 1, where H and L denote "high" (i.e., secret) and "low" (i.e., public) security variables or channels confidentiality. By observing the timing of the public channel L, an adversary can infer information about H. In order to prevent an attacker to infer the value of H, a countermeasure is to make sure that in any given system, the for-loop takes equal time for any two distinct values of H. This policy to defend against timing leaks is a *system property* (as opposed to the property of individual executions) and constitutes a *timed* hyperproperty. We propose here the temporal logic HyperMTL to express timed hyperproperties. Designing a temporal logic for timed hyperproperties has multiple challenges, and a trivial extension of HyperLTL to a timed logic or lifting MTL to a hyper logic result in a flawed or impractical framework. We discuss these issues next.

```
L := 1;
for (i := 1 to H) {do something};
L := 2;
```

Fig. 1. Timing leak.

Dealing with multiple timed traces. Timed traces of the same system may not *align*. Consider the following timed traces σ_1 and σ_2 shown on the left:

$$\sigma_1 = (\{a\},2)(\{b\},\ 5)(\{a,b\},8)\cdots \qquad \sigma'_1 = (\{a\},1)\ (\{a\},3)\ (\{a\},5)\cdots$$
$$\sigma_2 = (\{b\},3)(\{a,b\},4)(\{a\},\ 7)\cdots \qquad \sigma'_2 = (\{a\},1)\ (\{a\},2)\ (\{a\},3)\ (\{a\},4)(\{a\},5)\cdots$$

These traces have no matching time stamps, which makes reasoning about them simultaneously challenging. Another challenge is the speed of time progress in different traces. Consider traces σ'_1 and σ'_2 above on the right, where σ'_1 reaches time 5 in fewer steps than σ'_2, which has to be taken into account when reasoning about σ'_1 and σ'_2 simultaneously. These two examples show why HyperLTL cannot be trivially extended to a timed version, as the semantics of HyperLTL evaluates a set of traces *synchronously*, that is, positions advance in a lock-step manner (see Fig. 2 (left) for an example, where evaluation occurs at identical positions of both traces).

Decidability of verification. In order to allow reasoning about traces with different speeds, one has to allow *asynchronous* semantics, where one trace may make progress while another stutters (see Fig. 2 (right)). However, the computation power needed to reason about such a model of computation rises to the level required to solve the *post correspondance problem* (PCP) [32], which is known to be undecidable (the formal proof is out of the scope of this paper).

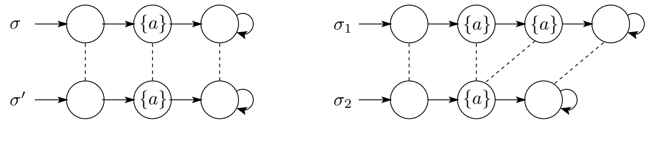

Fig. 2. Synchronous semantics (left) vs asynchronous semantics (right).

Our first contribution is the temporal logic HyperMTL with the following features:
— *Timed temporal operators.* HyperMTL generalizes HyperLTL by allowing explicit timing constraints over temporal operators. For example, formula $\varphi = \forall\pi.\forall\pi'.\Box_{[2,5]}(a_\pi \leftrightarrow a_{\pi'})$ means that any pair of traces π and π' should agree on the position of proposition a in all events within time period $[2,5]$. In addition to explicit timing constraints, we augment the temporal operators with features to express bounds on the difference in time elapse between traces. This is essential to realistically capture policies such as countermeasures to timing side-channels. For example, we enrich temporal operators to express properties like $\Diamond_{[0,\infty),[0,1]}(r_\pi \wedge r_{\pi'})$ which requires that proposition r should hold in events in trace π and π' that are at most one time unit of each other.

— *Two-layered design.* In order to obtain a decidable model-checking problem, HyperMTL is divided into two layers. The outer layer is interpreted with synchronous semantics, that is, the operators are evaluated position by position in a lock-step manner similar to HyperLTL (see Fig. 2 (left)). The inner layer allows temporal operators to be interpreted by the *asynchronous* semantics, where evaluation of traces can be asynchronous (see Fig. 2 (right)). For example, formula $\varphi = \forall\pi.\exists\pi'.\Box_{[0,\infty)}\mathsf{A}.(\Diamond_{[3,6],[1,2]}(r_\pi \wedge r_{\pi'}))$ means that for every trace π, there is another trace π', where it is (synchronously) always the case that eventually (asynchronously) within interval $[3,6]$ proposition r is observed in the two traces within interval $[1,2]$ of each other.

— *Framing.* Finally, the synchronous semantics can suffer from anomalies because evaluation points of a formula may depend on its context. We fix this problem by introducing the notion of formula *framing.* For instance, formulas *true* and $p_\pi \vee \neg p_\pi$ are only equivalent when *true* is evaluated at the ticking instants of π and therefore cannot be substituted as in an arbitrary context. Our solution allows to "frame" *true* and $p_\pi \vee \neg p_\pi$ forcing their evaluation to consider π (and only π) to regain substitutivity.

Our second contribution is to show the application of HyperMTL in different areas of computing. We demonstrate how HyperMTL can express important (1) security policies such as countermeasures to side-channel timing attacks and insecure composition, (2) service level agreements, and (3) properties of cyber-physical systems such as robustness, sensitivity, and overshoot observability.

Our third contribution is a model-checking algorithm for a fragment of HyperMTL. We show that the fragment with bounded intervals for the asynchronous operators and unrestricted synchronous operators the logic is decidable. This fragment covers all interesting examples listed above. We obtain the decidability result by reducing the model-checking problem for this fragment of HyperMTL to the model-checking of HyperLTL [13,24].

Organization The rest of the paper is organized as follows. In Section 2, we review the preliminary concepts. Then, we present the syntax and semantics of HyperMTL in Section 3, while its applications are discussed in Section 4. Our model checking algorithm is presented in Section 5. We discuss related work in Section 6 and finally conclude in Section 7.

2 Preliminaries

Let AP be a set of *atomic propositions* and $\Sigma = 2^{\mathsf{AP}}$ be the *alphabet*. We call each element of Σ a *letter*. A *trace* is an infinite sequence $\sigma = a_0 a_1 \cdots$ of letters from Σ. We use $\sigma(i)$ for a_i and σ^i for the suffix $a_i a_{i+1} \cdots$. An *indexed trace* is a pair (σ, p), where $p \in \mathbb{N}$ is a natural number (called the *pointer*). Indexed traces are used to traverse a trace by moving the pointer. Given an indexed trace (σ, p) and $n > 0$, we use $(\sigma, p) + n$ to denote the resulting indexed trace $(\sigma, p + n)$.

We fix the time domain to be non-negative integers $\mathbb{Z}_{\geq 0}$. An *event* is a pair (a, t), where $a \in \Sigma$ and $t \in \mathbb{Z}_{\geq 0}$. Given an event $e = (a, t)$, we use *label*(e) for a and *time*(e) for t. A *timed trace* is an infinite sequence $\sigma = (a_0, t_0)(a_1, t_1)(a_2, t_2) \ldots$, over $(\Sigma \times \mathbb{Z}_{\geq 0})$, such that for all $i \geq 0$, we have $t_i < t_{i+1}$. Given an indexed timed trace (σ, p), we use *time*(σ, p) to denote *time*$(\sigma(p))$.

2.1 HyperLTL

$\mathsf{HyperLTL}$ [13] is a temporal logic for hyperproperties, which allows reasoning about multiple execution traces simultaneously. The syntax of $\mathsf{HyperLTL}$ is:

$$\alpha ::= \exists \pi . \alpha \;\big|\; \forall \pi . \alpha \;\big|\; \varphi \qquad\qquad \varphi ::= a_\pi \;\big|\; \varphi \vee \varphi \;\big|\; \neg \varphi \;\big|\; \bigcirc \varphi \;\big|\; \varphi \,\mathcal{U}\, \varphi$$

where π is a *trace variable* from an infinite supply of trace variables. The intended meaning of a_π is that proposition $a \in \Sigma$ holds in the current time in trace π. Trace quantifiers $\exists \pi$ and $\forall \pi$ allow reasoning simultaneously about different traces of the computation. Atomic predicates a_π refer to a single trace π, and can be combined with Boolean operators to build relational tests as well as with temporal operators to construct temporal relational formulas. Informally, $\mathsf{HyperLTL}$ allows to reason about properties of systems that require to reason about the whole set of traces of the system at once, and not about each individual trace at a time.

Given a $\mathsf{HyperLTL}$ formula φ, we use $\mathit{Var}(\alpha)$ for the set of trace variables quantified in α. A formula α is well-formed if for all atoms a_π in α, π is quantified in α (i.e., $\pi \in \mathit{Var}(\alpha)$) and if no trace variable is quantified twice in α.

Given a set of traces W, the semantics of a $\mathsf{HyperLTL}$ formula α is defined in terms of trace assignments, which is a (partial) map from trace variables to indexed traces $\mathit{\Pi} : \mathit{Var}(\alpha) \rightharpoonup (W \times \mathbb{N})$. We use $\mathit{Dom}(\mathit{\Pi})$ for the subset of $\mathit{Var}(\alpha)$ for which $\mathit{\Pi}$ is defined. Given a trace assignment $\mathit{\Pi}$, a trace variable π, a trace σ and a pointer p, we denote by $\mathit{\Pi}[\pi \mapsto (\sigma, p)]$ the assignment that coincides with $\mathit{\Pi}$ for every path variable except for π, which is mapped to (σ, p). Also, we use $\mathit{\Pi} + n$ to denote trace assignment $\mathit{\Pi}'$ such that $\mathit{\Pi}'(\pi) = \mathit{\Pi}(\pi) + n$ for all $\pi \in \mathit{Dom}(\mathit{\Pi}) = \mathit{Dom}(\mathit{\Pi}')$. The semantics of $\mathsf{HyperLTL}$ is as follows:

$$\begin{array}{rll}
(W, \mathit{\Pi}) \models \exists \pi . \alpha & \text{iff} & \text{for some } \sigma \in W,\ (W, \mathit{\Pi}[\pi \mapsto (\sigma, 0)]) \models \alpha \\
(W, \mathit{\Pi}) \models \forall \pi . \alpha & \text{iff} & \text{for all } \sigma \in W,\ (W, \mathit{\Pi}[\pi \mapsto (\sigma, 0)]) \models \alpha \\
(W, \mathit{\Pi}) \models \varphi & \text{iff} & \mathit{\Pi} \models \varphi \\
\mathit{\Pi} \models a_\pi & \text{iff} & a \in \sigma(p),\ \text{where } (\sigma, p) = \mathit{\Pi}(\pi) \\
\mathit{\Pi} \models \varphi_1 \vee \varphi_2 & \text{iff} & \mathit{\Pi} \models \varphi_1 \text{ or } \mathit{\Pi} \models \varphi_2
\end{array}$$

$$\begin{array}{lll}
\Pi \models \neg\varphi & \text{iff} & \Pi \not\models \varphi \\
\Pi \models \bigcirc\varphi & \text{iff} & (\Pi + 1) \models \varphi \\
\Pi \models \varphi_1 \,\mathcal{U}\, \varphi_2 & \text{iff} & \text{for some } j \geq 0 \;\; (\Pi + j) \models \varphi_2 \\
 & & \quad \text{and for all } 0 \leq i < j, (\Pi + i) \models \varphi_1
\end{array}$$

Note that quantifiers assign traces to trace variables and set the pointer to the initial position 0. Also, the pointer in all trace assignments move in lock-step (at the same speed) within the semantics of $\mathcal{U}$ (that is, like in Fig. 2 (left), all pointers for different traces have the same value). Given a HyperLTL formula α and a Kripke structure that can generate a set of traces W, the model-checking problem for HyperLTL consists of deciding whether $(W, \Pi_\emptyset) \models \alpha$, where $\Pi_\emptyset$ is the trace assignment with $Dom(\Pi_\emptyset) = \emptyset$.

Example 1. The meaning of HyperLTL formula $\alpha = \forall\pi.\forall\pi'.\Box(a_\pi \leftrightarrow a_{\pi'})$ is that any pair of traces should agree on the value of a at every position.

In Sec 5, we will use a timed version of the $\mathcal{U}$ operator, so we define a derived temporal operator $\mathcal{U}_I$ that requires the satisfaction of the second argument in the interval I, using repeated applications of the next operator. The operator $\mathcal{U}_I$ is formally defined as follows ($\bigcirc^i$ refers $\bigcirc$ operator applied i times on the argument):

$$\varphi_1\mathcal{U}_{[a,b]}\varphi_2 \overset{\text{def}}{=} \bigvee_{a\leq i\leq b} (\bigcirc^i\varphi_2 \wedge \bigwedge_{0\leq j<i} \bigcirc^j\varphi_1), \quad \varphi_1\mathcal{U}_{[a,\infty)}\varphi_2 \overset{\text{def}}{=} \bigwedge_{0\leq i<a} \bigcirc^i\varphi_1 \wedge \bigcirc^a\varphi_1\mathcal{U}\varphi_2$$

2.2 Kripke Structures

We model timed systems as *timed Kripke structures* with time elapse in the arcs.

Definition 1. *A* Kripke structure *(KS) is a tuple* $\mathcal{M} = (S, S^0, \rightarrow, \mathsf{AP}, L)$*, where*
- S *is a set of* states*, and* $S^0 \subseteq S$ *is a set of* initial states*;*
- $\rightarrow \subseteq S \times \mathbb{Z}_{\geq 0} \times S$ *is a set of* transitions*;*
- AP *is the set of* atomic propostions*; and*
- $L : S \rightarrow 2^{\mathsf{AP}}$ *is a* labeling function *that assigns a set of atomic propositions to each state.*

We assume that every state has a successor, so each sink state s is equipped with a self-loop of the form $(s, 1, s)$. Note that untimed Kripke structures are simply Kripke structures for which all edges are of the form $(s, 1, s')$.

A *run* of a Kripke structure $(S, S^0, \rightarrow, \mathsf{AP}, L)$ from a state $s \in S$ is an infinite sequence of the form $\gamma_s = s_0d_0s_1d_1s_2d_2\cdots$, where $s_0 = s$ and for each $i \geq 0$, we have $(s_i, d_i, s_{i+1}) \in\rightarrow$, for some $d_i \in \mathbb{Z}_{\geq 0}$. A *trace* of a run γ_s is a timed trace of the form $(L(s_0), t_0)(L(s_1), t_1)(L(s_2), t_2)\cdots$, such that $t_0 = 0$ and for every $i > 0$, $t_{i+1} = t_i + d_i$, that is, we encode the delays from the actions of the Kripke structure into time-stamps on the letters of the timed trace. The *language* of a Kripke structure $\mathcal{M}$ (denoted $\mathcal{L}(\mathcal{M})$) is the set of all traces corresponding to runs of $\mathcal{M}$.

Example 2. The untimed interpretation of the Kripke structure $\mathcal{K}_1$ shown in Fig. 3 (left) satisfies HyperLTL formula $\alpha = \forall\pi.\forall\pi'.\Box(a_\pi \leftrightarrow a_{\pi'})$, whereas $\mathcal{K}_2$, in Fig. 3 (middle), does not.

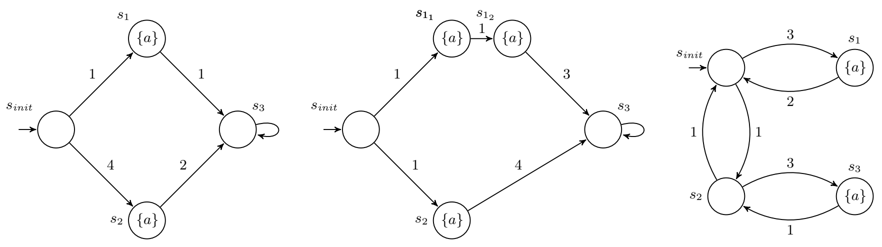

Fig. 3. Three timed Kripke structures: $\mathcal{K}_1$ (left) $\mathcal{K}_2$ (middle) and $\mathcal{K}_3$ (right).

3 The Temporal Logic HyperMTL

In this section, we present the syntax and semantics of HyperMTL. In HyperMTL, formulas are constructed in two *layers* using two kinds of temporal operators, *synchronous* and *asynchronous*. This layering enables the construction of both synchronous and asynchronous formulas, and a limited combination that still guarantees decidability. Synchronous operators allow comparing traces at the same point in time and reason about the time elapses. The evaluating time instants are those where at least one of the traces involved contains an event. If a trace σ does not contain an event at an evaluation time t then the closest previous event is used (we assume all traces contain an event at the initial time 0). Consider for example, traces σ and σ' shown on the right. The evaluation of $\{\sigma, \sigma'\}$ at time $t = 3$ considers event $(a_1, 1)$ for σ and $(b_2, 3)$ for σ'. On the other hand, asynchronous operators allow traces to proceed at different speeds and also allow to reason about the difference in elapsing times between two traces.

$$\sigma = (a_0,\, 0)\; (a_1,\, 1)\; (a_2,\, 4)\; (a_3,\, 5) \cdots$$
$$\sigma' = (b_0,\, 0)\; (b_1,\, 1)\; (b_2,\, 3)\; (b_3,\, 5) \cdots$$

We first present the syntax and then the semantics of each layer. Finally, we show how the choice of the evaluation point in the synchronous semantics can lead to unexpected logical cases and propose a simple *framing* mechanism to fix these anomalies.

3.1 Syntax

As with HyperLTL, the outermost layer (α formulas) introduces the trace quantifiers that bind trace variables; then the *synchronous layer* (φ formulas) introduces the synchronous temporal operators and the *asynchronous layer* (ψ formulas) introduce the asynchronous temporal constructs:

$$\begin{aligned}
\alpha &::= \exists\pi.\alpha \;\big|\; \forall\pi.\alpha \;\big|\; \varphi \\
\varphi &::= \mathit{true} \;\big|\; a_\pi \;\big|\; \varphi \vee \varphi \;\big|\; \neg\varphi \;\big|\; \varphi\, \mathcal{U}_I\, \varphi \;\big|\; \mathsf{A}.\psi \\
\psi &::= \mathit{true} \;\big|\; a_\pi \;\big|\; \psi \vee \psi \;\big|\; \neg\psi \;\big|\; \psi\, \mathcal{U}_{I,J}\, \psi
\end{aligned}$$

Here, $a \in \mathsf{AP}$ is an atomic proposition, and I and J are intervals of the form $[l, u]$ with $l, u \in \mathbb{Z}_{\geq 0} \cup \{\infty\}$ and $l \leq u$. As for HyperLTL, the meaning of a_π is that a

holds in the trace assigned to trace variable π. The intended meaning of $\varphi_1 \mathcal{U}_I \varphi_2$ is that there is an event at time t within an interval I, such that φ_2 holds at t and that φ_1 holds at all events before t. The meaning of interval $J = [l, u]$ is that the difference in time elapse between any two traces must be between l and u, when the obligation φ_2 is fulfilled. As usual, we use the syntactic sugar $\Diamond_I \varphi \overset{\text{def}}{=} \mathit{true}\, \mathcal{U}_I\, \varphi$ and $\Box_I \varphi \overset{\text{def}}{=} \neg(\Diamond_I \neg\varphi)$, and $\Diamond_{I,J} \varphi \overset{\text{def}}{=} \mathit{true}\, \mathcal{U}_{I,J}\, \varphi$, etc.

Temporal operators in synchronous formulas allow time to flow according to a global clock. The evolution of time in the evaluation of a formula proceeds according to the time-stamps of events in any of the traces in the formula. In case a_π is evaluated at a given time t, and the trace σ assigned to π does not contain an event at t, then the most recent past event in σ is used. The operator A (for (A)synchronous) denotes the evaluation of the subformula asynchronously, and allows to reuse temporal and propositional symbols. The asynchronous layer considers the possibility that traces proceed at different speeds which is captured by the notion of trajectory, also called a time-flow, defined below.

3.2 Semantics

The quantifier layer. To define the semantics of synchronous HyperMTL, we use pointed-timed trace assignments, that is, partial mappings $\mathit{\Pi}\colon \mathit{Var}(\alpha) \rightharpoonup (\Sigma \times \mathbb{Z}_{\geq 0})^\omega \times \mathbb{N}$, which are pointed assignments over timed traces. The intended meaning of $\mathit{\Pi}(\pi) = (\sigma, p)$ is that the event from timed trace σ at position p is currently used in the evaluation of trace π. Given a subset of trace variables $\overline{\pi} \subseteq \mathit{Var}(\alpha)$, we use $\mathit{\Pi} \setminus \overline{\pi}$ for the map that removes from the domain of $\mathit{\Pi}$ all $\pi \in \overline{\pi}$. As in the untimed case, $\mathit{\Pi}[\pi \mapsto (\sigma, p)]$ is the assignment that coincides with $\mathit{\Pi}$ for every trace variable except for π, which is mapped to (σ, p). The satisfaction of a HyperMTL formula φ over a timed word assignment $\mathit{\Pi}$, and a set W of timed words, denoted by $(W, \mathit{\Pi}) \models \varphi$, is defined as for HyperLTL:

$$\begin{array}{lll}
(W, \mathit{\Pi}) \models \exists \pi.\alpha & \text{iff} & (W, \mathit{\Pi}[\pi \mapsto (\sigma, 0)]) \models \alpha \text{ for some } \sigma \in W \\
(W, \mathit{\Pi}) \models \forall \pi.\alpha & \text{iff} & (W, \mathit{\Pi}[\pi \mapsto (\sigma, 0)]) \models \alpha \text{ for all } \sigma \in W \\
(W, \mathit{\Pi}) \models \varphi & \text{iff} & \mathit{\Pi} \models_s \varphi
\end{array}$$

where $\models_s$ is the semantics of the synchronous layer defined next.

The synchronous layer. We present first some preliminary definitions to capture the passage of time, by defining the sequence of *time ticks* for a given collection of traces. Intuitively speaking, this sequence contains the instants of the events in the union of events in the traces. We start by defining the position of trace σ at time t as the index of the latest event in σ whose time does not surpass t:

$$\mathit{pos}(\sigma, t) \overset{\text{def}}{=} i, \text{ such that } \mathit{time}(\sigma(i)) \leq t < \mathit{time}(\sigma(i+1))$$

Recall that $\mathit{time}(\sigma(i))$ is the time-stamp of the i-th event in σ. The position adjustment $\mathit{\Pi}|_t$ is an assignment (with $\mathit{Dom}(\mathit{\Pi}|_t) = \mathit{Dom}(\mathit{\Pi})$), which moves the pointer to the position denoted by t. That is, for $\pi \in \mathit{Dom}(\Pi)$, $\Pi|_t(\pi) = (\sigma, \mathit{pos}(\sigma, t))$ for $(\sigma, p) = \Pi(\pi)$. Then, given Π, we define the current instant and the next instant as follows:

$$\begin{aligned} \mathit{now}(\Pi) &= \max_{\pi \in \mathit{Dom}(\Pi)} \Big\{ \mathit{time}(\sigma(p)) \quad \mid \text{ for } (\sigma, p) = \Pi(\pi) \Big\} \\ \mathit{next}(\Pi) &= \min_{\pi \in \mathit{Dom}(\Pi)} \Big\{ \mathit{time}(\sigma(p+1)) \mid \text{ for } (\sigma, p) = \Pi(\pi) \Big\} \end{aligned}$$

Finally, the synchronous successor of Π is $\mathbf{succ}(\Pi) = \Pi|_{\mathit{next}(\Pi)}$. This can be extended to $\mathbf{succ}^{j+1}(\Pi) = \mathbf{succ}^j(\mathbf{succ}(\Pi))$. Note that, starting at time 0, **succ** follows the union of the time-stamps of $\mathit{Dom}(\Pi)$ in increasing order. We use $\Pi^{(j)}$ as short for $\mathbf{succ}^j(\Pi)$. We are now ready to define the semantics $\models_s$:

$$\begin{array}{lll} \Pi \models_s \mathit{true} & \text{iff} & \text{always holds} \\ \Pi \models_s a_\pi & \text{iff} & a \in \mathit{label}(\sigma(p)) \text{ for } (\sigma, p) = \Pi(\pi) \\ \Pi \models_s \varphi_1 \vee \varphi_2 & \text{iff} & \Pi \models_s \varphi_1 \text{ or } \Pi \models_s \varphi_2 \\ \Pi \models_s \neg\varphi & \text{iff} & \Pi \not\models_s \varphi \\ \Pi \models_s \varphi_1 \,\mathcal{U}_I\, \varphi_2 & \text{iff} & \text{for some } i \geq 0, \\ & & \quad \Pi^{(i)} \models_s \varphi_2 \text{ and } (\mathit{now}(\Pi^{(i)}) - \mathit{now}(\Pi)) \in I \\ & & \quad \text{and for all } j < i,\ \Pi^{(j)} \models_s \varphi_1 \\ \Pi \models_s \mathsf{A}.\psi & \text{iff} & \text{for some trajectory } \tau,\ (\Pi, \tau) \models_a \psi \end{array}$$

The notion of a *trajectory* is related to the asynchronous layer, defined next.

Example 3. Consider the following HyperMTL formula with only synchronous temporal operator $\alpha = \forall\pi.\forall.\pi'.\square_{[1,2]}(a_\pi \leftrightarrow a_{\pi'})$. The Kripke structure $\mathcal{K}_1$ in Fig. 3 (left) does not satisfy this formula, whereas $\mathcal{K}_2$ in Fig. 3 (middle) does.

The asynchronous layer. The asynchronous layer allows traces to proceed at different speeds. We start by the asynchronous semantics of intervals. Let α be a formula, $[l, u]$ be a time interval and Δ be a map from $\mathit{Var}(\alpha) \rightharpoonup \mathbb{Z}_{\geq 0}$ that gives a time duration for each π in $\mathit{Dom}(\Delta)$. We write $\Delta \models_I [l, u]$ whenever for all π in $\mathit{Dom}(\Delta)$, $\Delta(\pi) \in [l, u]$. We write $\Delta \models_J [l, u]$ whenever for all distinct $\pi, \pi' \in \mathit{Dom}(\Delta)$, $|\Delta(\pi) - \Delta(\pi')| \in [l, u]$.

A *trajectory* encodes which traces move and which traces stay at a given instant.

Definition 2. *Let V be a set of trace variables. A* trajectory *is an infinite sequence $\tau_0\tau_1\tau_2\cdots$ of subsets of V, such that for every $\pi \in V$, there are infinitely many $i \in \mathbb{Z}_{\geq 0}$ for which $\pi \in \tau_i$.*

For example, the trajectory depicted in Fig. 2 (right) is $\{\sigma_1, \sigma_2\}\{\sigma_1\}\{\sigma_1, \sigma_2\}\cdots$. Similar to timed traces, by τ^i, we mean the suffix $\tau_i\tau_{i+1}\ldots$. The asynchronous successor of a pointed time trace assignment Π with respect to a trajectory τ, denoted $\mathbf{succ}_a(\Pi, \tau)$, is the pair (Π', τ^1), where Π' is defined as follows. First, $\mathit{Dom}(\Pi') = \mathit{Dom}(\Pi)$; then for $\pi \in \mathit{Dom}(\pi')$, $\Pi'(\pi) = \Pi(\pi) + 1$ if $\pi \in \tau_0$, and $\Pi'(\pi) = \Pi(\pi)$ otherwise. Again, this definition can be extended to the j-th successor by defining $\mathbf{succ}_a^j$ to be the function that applies j times the function $\mathbf{succ}_a$. We use $(\Pi, \tau)^{(j)}$ as a short for $\mathbf{succ}_a^j(\Pi, \tau)$.

Given two pointed time trace assignments Π and Π' with the same domain $Dom(\Pi) = Dom(\Pi')$ (as it is the case with successors) the passage of time is defined as a map from $Var(\alpha) \rightharpoonup \mathbb{Z}_{\geq 0}$ that returns the passage of time for each assignment as follows:

$$\Delta(\Pi, \Pi')(\pi) \overset{\text{def}}{=} time(\Pi'(\pi)) - time(\Pi(\pi))$$

Finally, we define the time passage in j steps $\Delta^j(\Pi, \tau)$ as the time passage between the current evaluation instant and the evaluation instant obtained after j steps, that is, $\Delta^j(\Pi, \tau) = \Delta(\Pi, \Pi')$, where $(\Pi', \tau') = (\Pi, \tau)^{(j)}$. We are finally ready to define the semantics of the asynchronous layer:

$$
\begin{array}{lll}
(\Pi, \tau) \models_a true & \text{iff} & \text{always holds} \\
(\Pi, \tau) \models_a a_\pi & \text{iff} & a \in label(\sigma(p)) \text{ for } (\sigma, p) = \Pi(\pi) \\
(\Pi, \tau) \models_a \psi_1 \vee \psi_2 & \text{iff} & (\Pi, \tau) \models_a \psi_1 \text{ or } (\Pi, \tau) \models_a \psi_2 \\
(\Pi, \tau) \models_a \neg\psi & \text{iff} & (\Pi, \tau) \not\models_a \psi \\
(\Pi, \tau) \models_a \psi_1 \,\mathcal{U}_{I,J}\, \psi_2 & \text{iff} & \text{for some } i > 0, (\Pi, \tau)^{(i)} \models_a \psi_2, \\
 & & \quad \Delta^i(\Pi, \tau) \models_I I \text{ and } \Delta^i(\Pi, \tau) \models_J J, \text{ and} \\
 & & \quad \text{for all } j < i,\ (\Pi, \tau)^{(j)} \models_a \psi_1
\end{array}
$$

Essentially, the asynchronous until operator $\mathcal{U}_{I,J}$ checks whether following the trajectory τ the attempt ψ_2 is met (also satisfying the temporal constraints I and J) and the obligation ψ_1 is fulfilled at all previous evaluation instants.

Example 4. The meaning of formula $\alpha = \forall\pi.\exists\pi'.\Box_{[0,\infty)}\mathsf{A}.(\Diamond_{[0,\infty),[0,1]}(a_\pi \wedge a_{\pi'}))$ is that for every trace, there exists another one, such that it is (synchronously) always the case that proposition a is (asynchronously) observed in both traces within one time unit of each other. For example, $\mathcal{K}_3$ in Fig 3 (right) satisfies this formula.

3.3 Framing

Unfortunately, the semantics defined above has some unexpected drawbacks, which we fix here with the notion of framing. Consider a formula α that contains a synchronous sub-formula φ. The set of time instants at which φ is evaluated depends on the events of all traces in $Var(\alpha)$ (which is a super-set of the traces that actually appear as trace variables in φ). For example, consider the formula $\forall\pi.\exists\pi'.\big(\Diamond_{[5,10]}a_\pi \leftrightarrow \Diamond_{[4,10]}a_{\pi'}\big)$. The sub-formula $\varphi_1 = \Diamond_{[5,10]}a_\pi$ only refers to π while its enclosing formula refers to π and π'. Let σ and σ' be the traces assigned to π and π'. The semantics of the synchronous layer of HyperMTL evaluates φ_1 at the points at which either σ or σ' contain an event and not only at the points at which σ does. This evaluation causes semantic anomalies, illustrated in the following examples.

Example 5. Consider $\varphi = \Diamond_{[1,3]}true$ and $W = \{\sigma\}$ for $\sigma = (a, 0)(a, 2)(a, 4)\ldots$. In this case $(W, \Pi_\emptyset) \not\models \varphi$ but $(W, \Pi_\emptyset) \models \exists\pi.\varphi$ and $(W, \Pi_\emptyset) \models \forall\pi.\varphi$ even though $\pi \notin Var(\varphi)$.

In first-order logic, if a variable x does not appear in a formula P, P is equivalent to $\exists x.P$ and to $\forall x.P$, but the previous example illustrates that this is not the case for HyperMTL. It is true in the asynchronous semantics, but not necessarily for synchronous formulas due to the additional ticking instants.

Example 6. Consider formulas $\varphi_1 = \Diamond_{[1,2]} a_{\pi_1}$ and $\varphi_2 = \Diamond_{[3,4]} a_{\pi_2}$ and consider timed words $\sigma_1 = (a,0)(a,4)(a,8)\ldots$ and $\sigma_2 = (a,0)(a,2)(a,12)\ldots$ In this case $(\sigma_1, 0) \not\models \varphi_1$ and $(\sigma_2, 0) \not\models \varphi_2$, but $((\sigma_1,\sigma_2),0) \models \varphi_1 \wedge \varphi_2$. Even though σ_1 is the trace assigned to π_1, the ticks in σ_1 make φ_2 true (even though φ_2 does not refer to π_1 and hence "should not be affected" by the trace assigned to π_1).

To fix these anomalies, we enrich the synchronous layer with a *framing* operator $[\varphi]_{\overline{\pi}}$ that restricts the time-words that the semantics use, resulting in the following syntax:

$$\varphi ::= \mathit{true} \mid a_\pi \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi\, \mathcal{U}_I\, \varphi \mid \mathsf{A}.\psi \mid \ [\varphi]_{\overline{\pi}}$$

To be well-formed we require that every sub-formula of the form $[\varphi]_{\overline{\pi}}$ satisfies that $\mathit{Var}(\varphi) \subseteq \overline{\pi}$. We assume that every formula is well-formed. The semantics for the synchronous layer is now extended so $\Pi \models_s [\varphi]_{\overline{\pi}}$ whenever $\Pi[\overline{\pi}] \models_s \varphi$. Using this new definition we can prove that $[\mathit{true}]_\pi$ and $[p_\pi \vee \neg p_\pi]_\pi$ are equivalent, and can be substituted in every context (that includes π as a path).

4 HyperMTL in Action

We first note that in all the applications explored in this section, explicit framing was not necessary as the context of every sub-formula guarantee that it was evaluated when necessary.

Side-channel Timing Attacks. A timing attack is one that exploits the time-dependent behavioral characteristics of the implementation of an algorithm rather than other "functional" properties of the algorithm. For example, the execution time for the square-and-multiply algorithm used in modular exponentiation in encryption algorithms depends linearly on the number of '1' bits in the encryption key. While the number of '1' bits alone is not nearly enough information to make finding the key easily, repeated executions with the same key and different inputs can be used to perform statistical correlation analysis of timing information to recover the key completely, even by a passive attacker. This is a practical attack against a number of encryption algorithms, including RSA and ElGamal. In order to design a countermeasure against this attack, one can require that in any given system, for any pairs of executions, it should always be the case that, if the function is invoked in both executions, they both return within close enough times. The corresponding HyperMTL formula is $\varphi_{\mathsf{timing}}$ in Fig. 4, where inv and ret denote invocation and return of a function, interval $[0,10]$ is an upper bound on the execution time of the function, and interval $[0,1]$ specifies how close the execution times should be in π and π'.

$$\varphi_{\mathsf{timing}} = \forall\pi.\forall\pi'.\Box_{[0,\infty)}\mathsf{A}.\Big((\mathsf{inv}_\pi \wedge \mathsf{inv}_{\pi'}) \ \rightarrow\ \Diamond_{[0,10),[0,1]}(\mathsf{ret}_\pi \wedge \mathsf{ret}_{\pi'})\Big)$$

$$\varphi_{\mathsf{compose}} = \forall\pi.\forall\pi'.\Box_{[0,\infty)}\mathsf{A}.\Big((\mathsf{inv}^H_\pi \wedge \mathsf{inv}^H_{\pi'}) \ \rightarrow\ \Box_{[0,5],[0,2]}(\mathsf{hcopy}_\pi \wedge \mathsf{hcopy}_{\pi'})\Big)$$

$$\varphi_{\mathsf{sla}} = \forall\pi.\exists\pi'.\Box_{[0,\infty)}\mathsf{A}.\Big((\mathsf{req}_\pi \wedge \mathsf{req}_{\pi'}) \ \rightarrow\ \Diamond_{[0,100],[0,1]}(\mathsf{res}_\pi \wedge \mathsf{res}_{\pi'})\Big)$$

$$\varphi_{\mathsf{robust}} = \forall\pi.\forall\pi'.\Box_{[0,\infty)}\mathsf{A}.\Big(\Box_{I,[0,\infty)}\Big(d(x_\pi, x_{\pi'}) \leq c\Big) \ \rightarrow\ \Box_{I',[0,\infty)}\Big(d(y_\pi, y_{\pi'}) \leq c'\Big)\Big)$$

$$\varphi_{\mathsf{overshoot}} = \forall\pi.\forall\pi'.\Box_{[0,\infty)}\mathsf{A}.\Big[step_\pi \ \rightarrow\ \Box_{I,[0,\infty)}(x_\pi < c)\Big] \ \rightarrow$$
$$\Box_{[0,\infty)}\mathsf{A}.\Big[\big(step_{\pi'} \wedge \Diamond_{I,[0,\infty)}(x_{\pi'} > c)\big) \ \rightarrow \Diamond_{I,[0,\infty)} d(y_\pi, y_{\pi'}) > \epsilon\Big]$$

Fig. 4. Examples of properties expressed in HyperMTL.

Another example of timing leaks is related to composing secure components. For example, secure multi-execution (SME) [16] removes insecurities (including timing leaks) in any given process. To this end, it runs two copies, `H` (for secrets) and `L` (for public channels), of a given program; feeding (a copy of) `H` and `L` input to the `H`-copy, and dropping its `L` output; and feeding only `L` input to the `L`-copy, and dropping its `H` output. Since the `L`-copy receives no `H` input, no information can be leaked. In some implementations of SME inputs enter a queue, which is serviced by first running the `L`-copy on the `L` projection of the next input, then running the `H`-copy on the input. While this approach prevents leaks to output values, the time at which the `L`-copy processes the next input depends on how long it takes for the `H`-copy to finish processing previous inputs, which in turn, opens a timing channel. In other words, a correct composition of two secure components should satisfy $\varphi_{\mathsf{compose}}$ in Fig. 4. where inv^H denotes invocation of the `H`-copy and hcopy denotes that the execution is in the `H`-copy. Again, intervals $[0, 5]$ and $[0, 2]$ are arbitrary.

Service Level Agreements. A *service level agreement* (SLA) specifies acceptable performance of a system. These agreements often use statistics such as mean response time, the mean time that elapses between a request and a response. Other examples include time service factor, and percentage uptime. If these statistics are used to define policies across all executions of a system, then they are timed hyperproperties. Here, we consider a simple *fair* SLA policy, where no execution can be discriminated. More specifically, we require that for any execution with a particular response time, there has to exists another one with a similar timing behavior, as show in φ_{sla} in Fig. 4, where req and res denote request and response propositions, respectively, and intervals $[0, 100]$ and $[0, 1]$ are arbitrary.

Cyber-physical Systems. In cyber-physical systems, *robustness* is the ability of a computer system to cope with errors during execution and cope with erroneous input. More specifically, we require that if the distance between two different input values is bounded by some value c within some time interval I, then the distance between outputs is also bounded by some value c' within time interval I', as shown in $\varphi_{\mathsf{robust}}$ in Fig. 4, where x and y are input and output variables, respectively. Note that by abuse of notation x and y refer to the value of variables rather than atomic propositions.

Another feature in cyber-physical systems is *overshoot observability.* Here, we require that (1) in one execution, if a signal steps, then for some time interval I the input is bounded, (2) in another execution the signal steps and the input overshoots, then (3) the distance between the output signals is greater than some bound (i.e., the overshoot is observed in the output). This is shown in $\varphi_{\mathsf{overshoot}}$ in Fig. 4.

5 Model Checking

The model-checking problem is the following: Given a Kripke structure $\mathcal{K}$, whose language is W, and a HyperMTL formula α decide whether $(W, \Pi) \models \alpha$. In this section, we show that the model-checking problem for HyperMTL is decidable for a fragment of HyperMTL, where the intervals in the asynchronous until operator $\mathcal{U}_{I,J}$ are bounded. We will refer to this fragment as "bounded HyperMTL". Another fragment we will consider are formulas without the asynchronous sub-formulas (those starting with A). We will refer to the fragment as the "synchronous HyperMTL". Our approach to show decidability consists of the following:

- Step 1: Reduce the model-checking problem of bounded HyperMTL to that of synchronous HyperMTL.
- Step 2: Reduce the model-checking problem of synchronous HyperMTL to that of HyperLTL, which is known to be decidable [13].

We provide the details of these steps separately.

Bounded HyperMTL to Synchronous HyperMTL. Given a bounded HyperMTL formula α, we provide an algorithm to construct an equivalent synchronous HyperMTL formula $\hat{\alpha}$. The intuition is that an asynchronous formula $\mathsf{A}.\psi$ with bounded until operators only depends on a finite interval of a timed trace. Hence, the asynchronous formula can be replaced by a synchronous formula that encodes all finite interval patterns satisfied by $\mathsf{A}.\psi$.

First, we formalize when two timed traces σ and σ' agree on certain intervals. Given two timed traces σ and σ', and natural numbers r, r' and s, we say that σ and σ' are (r, r', s)-*conformant*, if the timed trace σ starting from r and the timed trace σ' starting from r' are the same for a duration of s, that is, for every i such that $r \leq \mathit{time}(\sigma(i)) \leq r+s$, there is a j such that $\mathit{time}(\sigma'(j)) - r' = \mathit{time}(\sigma(i)) - r$ and $\mathit{label}(\sigma(i)) = \mathit{label}(\sigma'(j))$ (and vice-versa for σ' and σ). The first time of an assignment defined as $\mathit{first}(\Pi) = \min_{\pi \in \mathit{Dom}(\Pi)} \Big\{ \mathit{time}(\sigma(p)) \mid \quad \text{for } (\sigma, p) = \Pi(\pi) \Big\}$.

This allows us to define conformance between assignments. Two assignments Π and Π' are *s-conformant*, if $Dom(\Pi) = Dom(\Pi')$ and for all $\pi \in Dom(\Pi)$, $label(\sigma(p)) = label(\sigma'(p'))$ and σ and σ' are $(first(\Pi), first(\Pi'), s)$*-conformant*, where $\Pi(\pi) = (\sigma, p)$ and $\Pi'(\pi) = (\sigma', p')$.

Next, given a bounded asynchronous formula ψ we define the future time period T_ψ of a timed trace which has an effect on the satisfaction of ψ. Let $ub(I)$ be the least upper-bound of an interval I. T_ψ is defined inductively as: $T_{true} = 0$; $T_{a_\pi} = 0$; $T_{\psi_1 \vee \psi_2} = \max\{T_{\psi_1}, T_{\psi_2}\}$; $T_{\neg\psi} = T_\psi$; $T_{\psi_1 \mathcal{U}_{I,J} \psi_2} = ub(I) + ub(J) + \max\{T_{\psi_1}, T_{\psi_2}\}$.

The next proposition formalizes the intuition that the satisfaction of a bounded asynchronous formula depends on only a finite interval of an assignment starting from the first time of the assignment.

Proposition 1. *Let ψ be bounded asynchronous, and let Π and Π' be two T_ψ-conformant assignments. Then, for any τ, $\Pi, \tau \models_a \psi$ if and only if $\Pi', \tau \models_a \psi$.*

The proof proceeds by induction on the structure of ψ. For the case $\psi = a_\pi$ the result follows because labels match at the pointer indices in the two assignments. For the until operator, $\psi_1 \mathcal{U}_{I,J} \psi_2$, a witness for ψ_2 happens at pointer values that satisfy I and J, hence, the latest pointer values corresponding to the witness are within $ub(I) + ub(J)$ from the $first(\Pi)$ and $first(\Pi')$, respectively.

Next, we provide a construction for a synchronous formula that encodes all the assignments that are conformant to a given assignment in a given interval. Let $\varphi^{s,\pi}_{\sigma,r}$ encode the pattern of σ in the interval $[r, s]$. More precisely, if $\sigma = (a_0, t_0)(a_1, t_1)\cdots$, then

$$\varphi^{s,\pi}_{\sigma,r} = \bigwedge_{i: r \leq time(\sigma(i)) \leq r+s, a = label(\sigma(i))} true\, \mathcal{U}_{[t_i, t_i]}\, a_\pi$$

Given any assignment Π and a natural number s, we construct a synchronous formula φ^s_Π such that $\Pi' \models_s \varphi^s_\Pi$ for every Π' that is s-conformant with Π.

$$\varphi^s_\Pi = \bigwedge_{\pi \in Dom(\Pi), \Pi(\pi) = (\sigma, p)} \left[\varphi^{s-(time(\sigma(p)) - first\Pi), \pi}_{\sigma, time(\sigma(p))}\right]_\pi .$$

Finally, we are ready to construct a synchronous formula $\hat{\psi}$ that is equivalent to a bounded synchronous formula $\mathsf{A}.\psi$. From Prop. 1, the satisfaction of ψ by an assignment Π only depends on the values in the interval $I = [first(\Pi), first(\Pi) + T_\psi]$. Given the values of an assignment Π in the interval I, one can algorithmically check if $\Pi \models_s \mathsf{A}.\psi$. More precisely, there are only finitely many τ's that are relevant within I, hence, by iterating over these τ's, and using the semantics of $\models_a$, one can effectively check $\Pi, \tau \models_a \psi$. Given a natural number s, let Π_s be the set of all assignments that are s-conformant with Π. Note that s-conformance is an equivalence relation on the set of assignments with finite index. Let $Rep(\Pi, s)$ denote a finite set of representative assignments for each equivalence class. Further, let $Sat(\psi, \Pi, s)$ denote those elements of $Rep(\Pi, s)$ that correspond to satisfying assignments for ψ. Then $\hat{\psi}$ is given by the disjunction of φ^s_Π for all $\Pi \in \mathit{Sat}(\psi, \Pi, s)$. Note that $\mathit{Sat}(\psi, \Pi, s)$ is computable, and hence, $\hat{\psi}$ (which does not contain asynchronous sub-formulas) is effectively constructible.

Lemma 1. *Given a bounded asynchronous formula ψ and an assignment Π, the $\Pi \models_s \mathsf{A}.\psi$ if and only if $\Pi \models_s \hat{\psi}$.*

Synchronous HyperMTL to HyperLTL. We show now how to transform a synchronous HyperMTL formula α to a HyperLTL formula $\hat{\alpha}$ such that the set of timed traces and timed assignments satisfying α is the same as the set of the corresponding untimed traces and assignments satisfying $\hat{\alpha}$. Then we reduce the model-checking problem of α with respect to a (timed) Kripke structure to that of $\hat{\alpha}$ with respect to an untimed Kripke structure.

Given a timed trace, its untiming refers to a sequence that contains an event at every time instant obtained by repeating an actual event until the next actual event in the time trace. Given a letter $a \in \Sigma$, let $\bar{a}$ be a fresh letter (not in Σ), used in the filled events between two actual occurrences. We also use the fresh special symbol $\epsilon \notin \Sigma$. Given a timed trace $\sigma = (a_0, t_0)(a_1, t_1)\cdots$, we define $\mathit{untime}(\sigma)$ to be the trace $b_0 b_1 \cdots$, where for each $j \geq 0$, $b_j = a_i$ if $j = t_i$ for some i, and $b_j = \bar{a}_i$ if $t_i < j < t_{i+1}$, and $b_j = \epsilon$ if $0 \leq j < t_0$. For instance, if $\sigma = (a,2)(b,5)(b,7)(a,9)\cdots$, $\mathit{untime}(\sigma) = \epsilon\epsilon a\bar{a}\bar{a}b\bar{b}b\bar{b}a\cdots$. For a Π, we define $\mathit{untime}(\Pi)$ to be the trace assignment, where $\mathit{Dom}(\Pi) = \mathit{Dom}(\mathit{untime}(\Pi))$ and for every $\pi \in \mathit{Dom}(\Pi)$, $\mathit{untime}(\Pi)(\pi) = (\mathit{untime}(\sigma), \mathit{now}(\Pi))$, where $\Pi(\pi) = (\sigma, p)$.

Next, given a synchronous HyperMTL formula φ, we define a transformed formula $U(\varphi)$ inductively as follows. Here, Event^φ corresponds to the occurrence of an event, and is defined as $\mathit{Event}^\varphi = \bigvee_{\pi \in \mathit{Var}(\varphi), a \in \mathsf{AP}} a_\pi$.

$$U(\mathit{true}) = \mathit{true} \qquad U(a_\pi) = a_\pi \vee \bar{a}_\pi$$
$$U(\varphi_1 \vee \varphi_2) = U(\varphi_1) \vee U(\varphi_2) \qquad U(\neg\varphi) = \neg U(\varphi)$$
$$U(\varphi_1 \,\mathcal{U}_I\, \varphi_2) = (\mathit{Event} \rightarrow U(\varphi_1))\,\mathcal{U}_I\,(\mathit{Event} \wedge U(\varphi_2)) \qquad U(Q.\alpha) = Q.U(\alpha)$$

Lemma 2. *Given a synchronous formula α, a set of timed traces W and an assignment Π, $(W, \Pi) \models \alpha$ if and only if $(\mathit{untime}(W), \mathit{untime}(\Pi)) \models U(\alpha)$.*

The above lemma can be proved by induction on the structure of α.

Given a Kripke structure $\mathcal{M}$ and a synchronous HyperMTL formula α, our objective is to check if $\mathcal{L}(\mathcal{M}), \Pi_\emptyset \models \alpha$. Lemma 2 states that it is equivalent to checking $\mathit{untime}(\mathcal{L}(\mathcal{M})), \mathit{untime}(\Pi_\emptyset) \models U(\alpha)$. It is straightforward to construct an $\hat{\mathcal{M}}$ that generates $\mathit{untime}(\mathcal{M})$ from $\mathcal{M}$ by replacing transitions in $\mathcal{M}$, say, from a state s to a state s' with delay d, by sequence of $d-1$ intermediate states whose labels are $\bar{a}$, where a is the label of s.

6 Related Work

There has been a lot of recent progress in automatically verifying [24,23,22,15] and monitoring [2,21,8,7,20,33,26] HyperLTL specifications, including a growing set of tools, like the model checker MCHyper [24,15], the satisfiability checkers EAHyper [19] and MGHyper [17], and the runtime monitoring tool RVHyper [20]. Synthesis techniques for HyperLTL has been studied in [18] and in [6].

Comparatively, much less attention has been put to timed hyperproperties. The work in [30] introduces HyperSTL, which extends STL by allowing quantification over real-valued signals, and proposes a monitoring algorithm for HyperSTL formulas. The work in [27] introduces the temporal logic timed HyperLTL, which adds one type of timing constraint to the until operator in the synchronous semantics of HyperLTL. This covers some timed hyperproperties, but it falls short in expressing requirements such as timing side-channels as presented in Section 4. Our formulation allows the execution times in different traces to be *similar* (i.e., $\Diamond_{[0,\infty),[0,1]}(r_\pi \wedge r_{\pi'})$), rather than just within a prescribed time bound as in [27]). Also, the proposed logic operates only in the HyperLTL synchronous semantics, meaning that all evaluations are conducted in the same trace positions.

Another recent work on timed hyperproperties is [28], which proposes an alternative definition to HyperMTL also distinguishing synchronous and asynchronous semantics, but there are fundamental differences. The synchronous semantics is similar to that of the one proposed in [27], and forces all traces to include events at the same instants, with the global time-stamp as an additional value. The asynchronous semantics in [27] is similar to our synchronous semantics, which keeps a global clock in the evaluation and proceeds in a total order. In comparison, our asynchronous semantics is based on the existence of a trajectory and allows to compare traces that evolve at different speeds, which cannot be captured in [28]. Additionally, most of the fragments of the logic in [28] are undecidable. Finally, the logic in [28] does not incorporate framing and suffers from many logical anomalies. For example, $\forall \pi_b.(p_{\pi_b} \,\mathcal{U}\, q_{\pi_b})$ is not equivalent to $\forall \pi_a.\forall \pi_b.(p_{\pi_b} \,\mathcal{U}\, q_{\pi_b})$ in spite of π_a not occurring in the inner formula (see [28] p. 16:6). Also, it is possible in the logic in [28] that for a given model M and formula φ, neither $M \models \varphi$ nor $M \models \neg\varphi$. All these anomalies are fixed by framing introduced here, but the formal proof is out of the scope of this paper.

7 Conclusion and Future Work

We introduced the temporal logic HyperMTL for *timed hyperproperties*. Even though our logic can be easily extended to richer models of time, we described here a discrete-time domain that guarantees a decidable model-checking problem. We showed that HyperMTL can elegantly express important properties such as countermeasures to a rich class of side-channel timing attacks, SLA, and properties of CPS such as robustness and overshoot detectability. To automate the verification task, we proposed a model checking algorithm by reducing the problem to model checking HyperLTL. As future work, we plan to implement our algorithm, build tools, and conduct case studies in the areas mentioned in Section 4. Other important research directions include foundational problems such as satisfiability, verification, monitoring, and synthesis for different fragments of HyperMTL, as well as extensions to richer time domains.

References

1. E. Ábrahám and B. Bonakdarpour. HyperPCTL: A temporal logic for probabilistic hyperproperties. In *Proceedings of the 15th International Conference on Quantitative Evaluation of Systems (QEST)*, pages 20–35, 2018.
2. S. Agrawal and B. Bonakdarpour. Runtime verification of k-safety hyperproperties in HyperLTL. In *Proceedings of the IEEE 29th Computer Security Foundations (CSF)*, pages 239–252, 2016.
3. B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.
4. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
5. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without bdds. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, pages 193–207, 1999.
6. B. Bonakdarpour and B. Finkbeiner. Program repair for hyperproperties. In *Proceedings of the 17th Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 423–441, 2019.
7. B. Bonakdarpour, C. Sánchez, and G. Schneider. Monitoring hyperproperties by combining static analysis and runtime verification. In *Proceedings of the 8th Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, pages 8–27, 2018.
8. N. Brett, U. Siddique, and B. Bonakdarpour. Rewriting-based runtime verification for alternation-free HyperLTL. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 77–93, 2017.
9. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.
10. A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: A new symbolic model checker. *Software Tools for Technology Transfer (STTT)*, 2(4):410–425, 2000.
11. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs*, pages 52–71, 1981.
12. E. M. Clarke, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. In *Computer Aided Verification (CAV)*, pages 450–462, 1993.
13. M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez. Temporal logics for hyperproperties. In *Proceedings of the 3rd Conference on Principles of Security and Trust POST*, pages 265–284, 2014.
14. M. R. Clarkson and F. B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.
15. N. Coenen, B. Finkbeiner, C. Sánchez, and L. Tentrup. Verifying hyperliveness. In *Proceedings of the 31st International Conference on Computer Aided Verification (CAV)*, pages 121–139, 2019.
16. Devriese D and F. Piessens. Noninterference through secure multi-execution. In *Proceedings of the 31st IEEE Symposium on Security and Privacy, S&P*, pages 109–124, 2010.
17. B. Finkbeiner, C. Hahn, and T. Hans. MGHyper: Checking satisfiability of HyperLTL formulas beyond the $\exists^*\forall^*$ fragment. In *Proceedings of the 16th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 521–527, 2018.

18. B. Finkbeiner, C. Hahn, P. Lukert, M. Stenger, and L. Tentrup. Synthesizing reactive systems from hyperproperties. In *Proceedings of the 30th International Conference on Computer Aided Verification (CAV)*, pages 289–306, 2018.
19. B. Finkbeiner, C. Hahn, and M. Stenger. Eahyper: Satisfiability, implication, and equivalence checking of hyperproperties. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV)*, pages 564–570, 2017.
20. B. Finkbeiner, C. Hahn, M. Stenger, and L. Tentrup. RVHyper: A runtime verification tool for temporal hyperproperties. In *Proceedings of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 194–200, 2018.
21. B. Finkbeiner, C. Hahn, M. Stenger, and L. Tentrup. Monitoring hyperproperties. *Formal Methods in System Design (FMSD)*, 54(3):336–363, 2019.
22. B. Finkbeiner, C. Hahn, and H. Torfah. Model checking quantitative hyperproperties. In *Proceedings of the 30th International Conference on Computer Aided Verification*, pages 144–163, 2018.
23. B. Finkbeiner, Ch. Müller, H. Seidl, and E. Zalinescu. Verifying Security Policies in Multi-agent Workflows with Loops. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*, 2017.
24. B. Finkbeiner, M. N. Rabe, and C. Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In *Proceedings of the 27th International Conference on Computer Aided Verification (CAV)*, pages 30–48, 2015.
25. J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symp. on Security and Privacy*, pages 11–20, 1982.
26. C. Hahn, M. Stenger, and L. Tentrup. Constraint-based monitoring of hyperproperties. In *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 115–131, 2019.
27. J. Heinen. Model checking timed hyperproperties. Master's thesis, Saarland University, 2018.
28. H.-M. Ho, R. Zhou, and T. M. Jones. On verifying timed hyperproperties. In *Proceedings of the 26th International Symposium on Temporal Representation and Reasoning (TIME)*, pages 20:1–20:18, 2019.
29. G. Holzmann. The model checker spin. *IEEE Transactions on Software Engineering*, 1997.
30. L. V. Nguyen, J. Kapinski, X. Jin, J. V. Deshmukh, and T. T. Johnson. Hyperproperties of real-valued signals. In *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, pages 104–113, 2017.
31. A. Pnueli. The temporal logic of programs. In *Symposium on Foundations of Computer Science (FOCS)*, pages 46–57, 1977.
32. M. Sipser. *Introduction to the Theory of Computation.* Cengage Learning, 3rd edition, 2012.
33. S. Stucki, C. Sánchez, G. Schneider, and B. Bonakdarpour. Graybox monitoring of hyperproperties. In *Proceedings of the 23rd International Symposium on Formal Methods (FM)*, 2019. To appear.
34. M.Y. Vardi and P. Wolper. Automata theoretic techniques for modal logic of programs. *Journal of Computer and System Sciences*, 32:183–221, 1986.
35. S. Zdancewic and A. C. Myers. Observational determinism for concurrent program security. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW)*, page 29, 2003.