On Verifying Timed Hyperproperties

Hsi-Ming Ho
University of Cambridge, UK
hsi-ming.ho@cl.cam.ac.uk

Ruoyu Zhou
University of Cambridge, UK
ruoyu.zhou@cl.cam.ac.uk

Timothy M. Jones
University of Cambridge, UK
timothy.jones@cl.cam.ac.uk

Abstract

We study the satisfiability and model-checking problems for *timed hyperproperties* specified with HyperMTL, a timed extension of HyperLTL. Depending on whether interleaving of events in different traces is allowed, two possible semantics can be defined for timed hyperproperties: *synchronous* and *asynchronous*. While the satisfiability problem can be decided similarly as for HyperLTL regardless of the choice of semantics, we show that the model-checking problem for HyperMTL, unless the specification is alternation-free, is undecidable even when very restricted timing constraints are allowed. On the positive side, we show that model checking HyperMTL with quantifier alternations is possible under certain conditions in the synchronous semantics, or when there is a fixed bound on the length of the time domain.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Timed Automata; Temporal Logics; Cybersecurity

Digital Object Identifier 10.4230/LIPIcs.TIME.2019.16

Related Version A full version of the paper is available at `http://arxiv.org/pdf/1812.10005`.

Funding This work was supported by the Engineering and Physical Sciences Research Council (EPSRC), through grant references EP/K026399/1 and EP/P020011/1.

1 Introduction

Background. One of the most popular specification formalisms for reactive systems is *Linear Temporal Logic* (LTL), first introduced into computer science by Pnueli [52] in the late 1970s. The success of LTL can be attributed to the fact that its satisfiability and model-checking problems are of lower complexity (PSPACE-complete, as compared with non-elementary for the equally expressive first-order logic of order) and it enjoys simple translations into automata and excellent tool support (e.g., [15, 35]).

While LTL is adequate for describing features of *individual* execution traces, many security policies in practice are based on relations between *two (or more)* execution traces. A standard example of such properties is *observational determinism* [37, 54, 59]: for every pair of execution traces, if the low-security inputs agree in both execution traces, then the low-security outputs in both execution traces must agree as well. Such properties are called *hyperproperties* [17]: a model of the property is not a single execution trace but a set of execution traces. HyperLTL [16], obtained from LTL by adding *trace quantifiers*, has been proposed as a specification formalism to express hyperproperties. For example, operational determinism can be expressed as the HyperLTL formula:

$$\forall \pi_a \, \forall \pi_b \; \mathbf{G}(I_a = I_b) \Rightarrow \mathbf{G}(O_a = O_b) \, .$$

HyperLTL inherits almost all the benefits of LTL; in particular, tools that support HyperLTL verification can be built by leveraging existing tools for LTL.

For many applications, however, in addition to the occurences and orders of events, *timing* has to be accounted for as well. For example, one may want to verify that in every execution trace of the system, whenever a request `req` is issued, the corresponding acknowledgement `ack` is received within the next 5 time units. *Timed automata* [4] and *timed logics* [5, 9, 39] are introduced exactly for this purpose. In the context of security, timing anomalies caused by different high-security inputs is a realistic attack vector that can be exploited to obtain sensitive information; this kind of *timing side-channel attacks* also play significant roles in high-profile exploits like Meltdown [45] and Spectre [38]. In order to detect such undesired characteristics of systems, one needs to reason about *timed hyperproperties*.

▶ **Example 1** ([55]). A piece of C code that selects between two variables `x` and `y` based on a secret selection bit `b` (i.e. the user gets the output—either `x` or `y`—but does not know which one was actually selected) may be written as follows:

```
uint32_t select_u32(uint32_t b, uint32_t x, uint32_t y)
{
    return b ? x : y;
}
```

This straightforward implementation, however, may result in a timing side channel—depending on what compiler optimisations are applied, the execution time can depend on which of `x` and `y` is returned. In sensitive applications like cryptography libraries and embedded smart-card software, such code snippets are usually replaced by obfuscated, functional-equivalent versions, with the hope of eliminating the potential leakage of secret information. In this case, one such version is as follows:

```
uint32_t ct_select_u32(uint32_t b, uint32_t x, uint32_t y)
{
    signed bit = 0 - b;
    return (x & bit) | (y & ~bit);
}
```

Nevertheless, such attempts of obfuscation can easily be wiped out by more agressive code optimisations. For instance, after compilation by `clang 3.3` (`-O2`), the C code above results in the following assembly code, which contains a jump instruction and may still reveal the truth value of `b` via differences in execution times due to branch prediction. The issue can, however, be detected by an analysis based on suitable instruction-level timing models.

```
ct_select_u32:
    mov    0x4(%esp),%al
    test   %al,%al
    jne    L
    lea    0xc(%esp),%eax
    mov    (%eax),%eax
    ret
L:  lea    0x8(%esp),%eax
    mov    (%eax),%eax
    ret
```

Given the highly-sophisticated cache hierarchies, pipeline stalls, etc. in contemporary real machines, the timing side channel in the example above may be difficult to realise and exploit in an actual attack; but such issues may also manifest themselves at lower levels (e.g., RTL), as illustrated by the following example.

▶ **Example 2** ([44]). An AND gate with two inputs A, B and an output C and respective delays T_A, T_B, and T_C can be modelled as the timed automaton with two clocks x, y

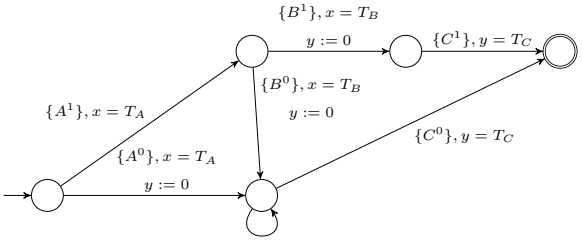


Figure 1 A timed automaton modelling an AND gate with inputs A, B and output C with respective delays T_A, T_B, and T_C.

in Figure 1 where $x = T_A$ checks if the value of clock x is T_A, $y := 0$ resets clock y to 0, etc. (suppose that $T_A < T_B$ and $T_B - T_A < T_C$). Intuitively, the truth values of A and B are obtained after T_A and T_B respectively, and the output $C = A \wedge B$ has a delay of T_C from the point when its value is confirmed. Of course, once A turned out to be 0 (i.e. A^0 has happened), the output C must be 0 as well. But the *time* C^0 happens (assuming $C = 0$) also depends on the truth value of A. In other words, when $C = 0$, a low-security user (to whom A^0 and A^1 are non-observable), provided that he/she can measure time, can also infer the truth value of A while he/she should not be able to. The pair of traces with $C = 0$ that reveals A is depicted in Figure 2 and Figure 3. In this simple example, however, the timing side channel can be removed by adding $y := 0$ on the self-loop on the lower-right location.

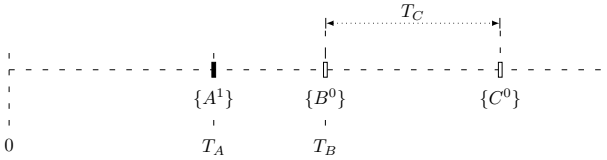


Figure 2 A trace ρ_1 with $A = 1$, $B = 0$, and $C = 0$.

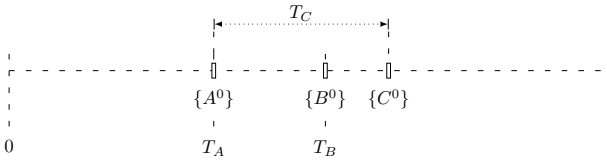


Figure 3 A trace ρ_2 with $A = 0$, $B = 0$, and $C = 0$.

Contributions. We propose HyperMTL, obtained by adding trace quantifiers to *Metric Temporal Logic* (MTL) [39], as a specification formalism for timed hyperproperties. We consider systems modelled as *timed automata*, and thus system behaviours are sequences of *events* that happen at different instants in time; this gives two possible pointwise semantics of HyperMTL: *asynchronous* and *synchronous* (this is in contrast to HyperLTL, for which a synchronous semantics is sufficient). We show that, as far as satisfiability is concerned, HyperMTL is similar to HyperLTL, i.e. satisfiability is decidable for fragments not containing $\forall\exists$, regardless of which semantics is assumed. However, in contrast with HyperLTL (whose model-checking problem is decidable), model checking HyperMTL is undecidable if there is at least one quantifier alternation in the specification, even when the timing constraints used in either the system or the specification are very restricted. Still, the alternation-free fragment of HyperMTL, which is arguably sufficient to capture many timed hyperproperties of practical interest, has a decidable model-checking problem. Finally, we identify several

subcases where HyperMTL model checking is decidable for larger fragments, such as when the synchronous semantics is assumed, the model is untimed, and the specification belongs to a certain subclass of one-clock timed automata, or when the time domain is bounded *a priori* by some $N \in \mathbb{N}_{>0}$.

Related work. Since the pioneering work of Clarkson and Schneider [17], there has been great interest in specifying and verifying hyperproperties in the past few years. The framework based on HyperLTL [16] is possibly the most popular for this purpose, thanks to its expressiveness, flexibility, and relative ease of implementation. In addition to satisfiability [23, 24] and model checking [16, 28], tools for monitoring HyperLTL also exist [3, 25, 26]. Notably, the complexity of monitoring HyperLTL, as well as model checking HyperLTL on restricted (tree-shaped or acyclic) Kripke structures, are studied in [12] and shown to be much lower than those of the general satisfiability and model-checking problems. These results, however, do not apply in the current timed setting—we will see in Section 4 that our main undecidability result holds even with these structural restrictions on the system.

Our formulation of HyperMTL is very closely related to HyperSTL [47] originally proposed in the context of quality assurance of cyber-physical systems. While [47] focusses on testing, we are concerned with the decidability of verification problems. On the other hand, the semantics of HyperSTL is defined over sets of continuous signals, i.e. state-based; as noted in [47], however, the price to pay for the extra generality is that implementing a model checker for HyperSTL is very difficult, especially for systems modelled in proprietary frameworks (such as Simulink®). Practical reasoning of HyperMTL, by contrast, can be carried out easily with existing highly optimised timed automata verification back ends, e.g., Uppaal [43].[1] Indeed, a prototype model checker based on Uppaal for the synchronous semantics of HyperMTL (with some restrictions) is reported in [32], although it does not consider the decidability of verification problems. Another relevant work [30], also based on Uppaal, checks noninterference in systems modelled as timed automata (similar to Example 4; see below). Their approach, however, is specifically tailored to noninterference and does not generalise. Some similar (but different) notions of noninterference for timed automata have been considered in [29, 58].

It is also possible to extend hyperlogics in other quantitative dimensions orthogonal to time. HyperPCTL [2] can express *probabilisitic hyperproperties*, e.g., the probability distribution of the low-security outputs are independent of the high-security inputs. In [27], specialised algorithms are developed for verifying *quantitative hyperproperties*, e.g., there is a bound on the number of traces with the same low-security inputs but different low-level outputs. The current paper is complementary to these works.

2 Timed hyperproperties

Timed words. A *timed word* (or a *trace*) over a finite alphabet Σ is a finite sequence of *events* $(\sigma_1, \tau_1) \dots (\sigma_n, \tau_n) \in (\Sigma \times \mathbb{R}_{\geq 0})^*$ with $\tau_1 \dots \tau_n$ an increasing sequence of non-negative real numbers ('*timestamps*'), i.e. $\tau_i < \tau_{i+1}$ for all i, $1 \leq i < n$.[2] For $t \in \mathbb{R}_{\geq 0}$ and a timed

[1] For more detailed accounts of the state-based and event-based semantics for timed automata and logics, see, e.g., [7, 51].

[2] To simplify the exposition, we focus on *finite* timed words in this paper (this assumption does not make the verification problems easier in general; e.g., HyperLTL satisfiability remains undecidable). All of our technical results carry over to the case of infinite timed words with some simple modifications. For example, in Section 3, suitable subformulae can be added to rule out the runs that get stuck in

word $\rho = (\sigma_1, \tau_1)\dots(\sigma_n, \tau_n)$, we write $t \in \rho$ iff $t = \tau_i$ for some i, $1 \leq i \leq n$. We denote by $T\Sigma^*$ the set of all timed words over Σ. A *timed language* (or a *trace property*) is a subset of $T\Sigma^*$.

Timed automata. Let X be a finite set of *clocks* ($\mathbb{R}_{\geq 0}$-valued variables). A *valuation* v for X maps each clock $x \in X$ to a value in $\mathbb{R}_{\geq 0}$. The set $G(X)$ of *clock constraints* (*guards*) g over X is generated by $g := \top \mid g \wedge g \mid x \bowtie c$ where $\bowtie \in \{\leq, <, \geq, >\}$, $x \in X$, and $c \in \mathbb{N}_{\geq 0}$. The satisfaction of a guard g by a valuation v (written $v \models g$) is defined in the usual way. For $t \in \mathbb{R}_{\geq 0}$, we let $v + t$ be the valuation defined by $(v + t)(x) = v(x) + t$ for all $x \in X$. For $\lambda \subseteq X$, we let $v[\lambda \leftarrow 0]$ be the valuation defined by $(v[\lambda \leftarrow 0])(x) = 0$ if $x \in \lambda$, and $(v[\lambda \leftarrow 0])(x) = v(x)$ otherwise.

A *timed automaton* (TA) over Σ is a tuple $\mathcal{A} = \langle \Sigma, S, s_0, X, \Delta, F \rangle$ where S is a finite set of locations, $s_0 \in S$ is the initial location, X is a finite set of clocks, $\Delta \subseteq S \times \Sigma \times G(X) \times 2^X \times S$ is the transition relation, and F is the set of accepting locations. We say that $\mathcal{A}$ is *deterministic* iff for each $s \in S$ and $\sigma \in \Sigma$ and every distinct pair of transitions $(s, \sigma, g^1, \lambda^1, s^1) \in \Delta$ and $(s, \sigma, g^2, \lambda^2, s^2) \in \Delta$, $g^1 \wedge g^2$ is not satisfiable. A *state* of $\mathcal{A}$ is a pair (s, v) of a location $s \in S$ and a valuation v for X. A *run* of $\mathcal{A}$ on a timed word $(\sigma_1, \tau_1)\dots(\sigma_n, \tau_n) \in T\Sigma^*$ is a sequence of states $(s_0, v_0)\dots(s_n, v_n)$ where (i) $v_0(x) = 0$ for all $x \in X$ and (ii) for each i, $0 \leq i < n$, there is a transition $(s_i, \sigma_{i+1}, g, \lambda, s_{i+1})$ such that $v_i + (\tau_{i+1} - \tau_i) \models g$ (let $\tau_0 = 0$) and $v_{i+1} = \big(v_i + (\tau_{i+1} - \tau_i)\big)[\lambda \leftarrow 0]$. A run of $\mathcal{A}$ is *accepting* iff it ends in a state (s, v) with $s \in F$. A timed word is *accepted* by $\mathcal{A}$ iff $\mathcal{A}$ has an accepting run on it. We denote by $[\![\mathcal{A}]\!]$ the timed language of $\mathcal{A}$, i.e. the set of all timed words accepted by $\mathcal{A}$. Two fundamental results on TAs are that the *emptiness* problem is decidable (PSPACE-complete), but the *universality* problem is undecidable [4].

Timed logics. The set of MTL formulae over a finite set of atomic propositions AP is generated by

$$\psi := \top \mid p \mid \psi_1 \wedge \psi_2 \mid \neg\psi \mid \psi_1 \,\overline{\mathbf{U}}_I\, \psi_2 \mid \psi_1 \,\overline{\mathbf{S}}_I\, \psi_2$$

where $p \in \mathsf{AP}$ and $I \subseteq \mathbb{R}_{\geq 0}$ is a *non-singular* interval with endpoints in $\mathbb{N}_{\geq 0} \cup \{\infty\}$.[3] We omit the subscript I when $I = [0, \infty)$ and sometimes write pseudo-arithmetic expressions for constraining intervals, e.g., '< 3' for $[0, 3)$. The other Boolean operators are defined as usual: $\bot \equiv \neg\top$ and $\psi_1 \vee \psi_2 \equiv \neg(\neg\psi_1 \wedge \neg\psi_2)$. We also define the dual temporal operators $\psi_1 \widetilde{\mathbf{U}}_I \psi_2 \equiv \neg\big((\neg\psi_1)\,\overline{\mathbf{U}}_I\,(\neg\psi_2)\big)$ and $\psi_1 \widetilde{\mathbf{S}}_I \psi_2 \equiv \neg\big((\neg\psi_1)\,\overline{\mathbf{S}}_I\,(\neg\psi_2)\big)$. Using these operators, every MTL formula ψ can be transformed into an MTL formula in *negative normal form*, i.e. $\neg$ is only applied to atomic propositions. To ease the presentation, we will also use the usual shortcuts like $\overline{\mathbf{F}}_I\,\psi \equiv \top\,\overline{\mathbf{U}}_I\,\psi$, $\overline{\mathbf{G}}_I\,\psi \equiv \neg\,\overline{\mathbf{F}}_I\,\neg\psi$, $\mathbf{X}_I\,\psi \equiv \bot\,\overline{\mathbf{U}}_I\,\psi$, and 'weak-future' variants of temporal operators, e.g., $\mathbf{F}\,\psi \equiv \psi \vee \overline{\mathbf{F}}\,\psi$. Given an MTL formula ψ over AP in negative normal form, a timed word $\rho = (\sigma_1, \tau_1)\dots(\sigma_n, \tau_n)$ over $\Sigma_{\mathsf{AP}} = 2^{\mathsf{AP}}$, and $t \in \mathbb{R}_{\geq 0}$, we define the MTL satisfaction relation $\models$ as follows:[4]

- $(\rho, t) \models \top$ iff $t \in \rho$;
- $(\rho, t) \models \bot$ iff $t \notin \rho$;

self-loops labelled with $\{p^e\}$.

[3] In the literature, this logic (with the requirement that constraining intervals must be non-singular) is usually referred to as MITL [5], but we simply call it MTL in this paper for notational simplicity. Also note that our undecidability results carry over to the fragment with only future operators.

[4] The formulation of the pointwise semantics of MTL here deviates slightly from the standard one (cf. [8, 50]) to enable a formal treatment of interleaving of events in different traces.

- $(\rho, t) \models p$ iff $t \in \rho$ and $p \in \sigma_i$;
- $(\rho, t) \models \neg p$ iff $t \in \rho$ and $p \notin \sigma_i$;
- $(\rho, t) \models \psi_1 \wedge \psi_2$ iff $(\rho, t) \models \psi_1$ and $(\rho, t) \models \psi_2$;
- $(\rho, t) \models \psi_1 \vee \psi_2$ iff $(\rho, t) \models \psi_1$ or $(\rho, t) \models \psi_2$;
- $(\rho, t) \models \psi_1 \,\overline{\mathbf{U}}_I\, \psi_2$ iff there exists $t' > t$ such that $t' - t \in I$, $(\rho, t') \models \top$, $(\rho, t') \models \psi_2$, and $(\rho, t'') \models \psi_1$ for all t'' such that $t'' \in (t, t')$ and $(\rho, t'') \models \top$;
- $(\rho, t) \models \psi_1 \widetilde{\overline{\mathbf{U}}}_I \psi_2$ iff for all $t' > t$ such that $t' - t \in I$ and $(\rho, t') \models \top$, either $(\rho, t') \models \psi_2$ or $(\rho, t'') \models \psi_1$ for some t'' such that $t'' \in (t, t')$ and $(\rho, t'') \models \top$;
- $(\rho, t) \models \psi_1 \,\overline{\mathbf{S}}_I\, \psi_2$ iff there exists t', $0 \leq t' < t$ such that $t - t' \in I$, $(\rho, t') \models \top$, $(\rho, t') \models \psi_2$, and $(\rho, t'') \models \psi_1$ for all t'' such that $t'' \in (t', t)$ and $(\rho, t'') \models \top$;
- $(\rho, t) \models \psi_1 \widetilde{\overline{\mathbf{S}}}_I \psi_2$ iff for all t', $0 \leq t' < t$ such that $t - t' \in I$ and $(\rho, t') \models \top$, either $(\rho, t') \models \psi_2$ or $(\rho, t'') \models \psi_1$ for some t'' such that $t'' \in (t', t)$ and $(\rho, t'') \models \top$.

We say that ρ *satisfies* ψ ($\rho \models \psi$) iff $(\rho, 0) \models \psi$, and we write $[\![\psi]\!]$ for the timed language of ψ, i.e. the set of all timed words satisfying ψ. It is well known that any MTL formula can be translated into a TA accepting the same timed language [6]; this implies that the satisfiability and model-checking problems for MTL are decidable (EXPSPACE-complete).

Adding trace quantifiers. Let V be an infinite supply of *trace variables*, the set of HyperMTL formulae over AP are generated by

$$\varphi := \exists \pi\, \varphi \mid \forall \pi\, \varphi \mid \psi$$
$$\psi := \top \mid \top_\pi \mid p_\pi \mid \psi_1 \wedge \psi_2 \mid \neg\psi \mid \psi_1 \,\overline{\mathbf{U}}_I\, \psi_2 \mid \psi_1 \,\overline{\mathbf{S}}_I\, \psi_2$$

where $\pi \in V$, $p \in \mathsf{AP}$, and $I \subseteq \mathbb{R}_{\geq 0}$ is a non-singular interval with endpoints in $\mathbb{N}_{\geq 0} \cup \{\infty\}$ (to ease the notation, we will usually write, e.g., p_a for p_{π_a}). Without loss of generality we forbid the reuse of trace variables, i.e. each trace quantifier must use a fresh trace variable. Syntactic sugar is defined as in MTL, e.g., $\overline{\mathbf{F}}_I\, \psi \equiv \top \,\overline{\mathbf{U}}_I\, \psi$. A HyperMTL formula is *closed* if it does not have free occurrences of trace variables. Following [22], we refer to fragments of HyperMTL by their quantifier patterns, e.g., $\exists^*\forall^*$-HyperMTL. Finally, note that trace quantifiers can be added to TAs in the same manner (in this case, quantified TAs operate over 'stacked' traces; see the semantics for HyperMTL below).

In contrast with TAs and MTL formulae, which define *trace properties*, HyperMTL formulae define *(timed) hyperproperties*, i.e. sets of trace properties. Depending on whether one requires timestamps in quantified traces to match exactly (i.e. all quantified traces must *synchronise*), two possible semantics can be defined accordingly.

Asynchronous semantics. A *trace assignment* over Σ is a partial mapping from V to $T\Sigma^*$. We write $\Pi_\emptyset$ for the empty trace assignment and $\Pi[\pi \mapsto \rho]$ for the trace assignment that maps π to ρ and π' to $\Pi(\pi')$ for all $\pi' \neq \pi$. Given a HyperMTL formula φ over AP whose quantifier-free part is in negative normal form, a trace set T over Σ_{AP}, a trace assignment Π over Σ_{AP}, and $t \in \mathbb{R}_{\geq 0}$, we define the HyperMTL *asynchronous* satisfaction relation $\models$ as follows (we omit the cases where the definitions are obvious or exactly similar):

- $(T, t) \models_\Pi \top$ iff $t \in \rho$ for some $\rho \in \mathit{range}(\Pi)$;[5]
- $(T, t) \models_\Pi \top_\pi$ iff $t \in \rho$ for $\rho = \Pi(\pi)$;
- $(T, t) \models_\Pi p_\pi$ iff $t \in \rho$ for $\rho = \Pi(\pi)$ and $p \in \sigma_i$ for the event (σ_i, t) in ρ;

[5] Note the dependency of the interpretation of $\top$ on Π; in particular, it is possible for a trace set with out-of-sync traces to satisfy $\forall \pi_b\, (p_b \,\overline{\mathbf{U}}\, q_b)$ but not $\forall \pi_a \forall \pi_b\, (p_b \,\overline{\mathbf{U}}\, q_b)$.

- $(T,t) \models_\Pi \psi_1 \,\overline{\mathbf{U}}_I\, \psi_2$ iff there exists $t' > t$ such that $t' - t \in I$, $(T,t') \models_\Pi \top$, $(T,t') \models_\Pi \psi_2$, and $(T,t'') \models_\Pi \psi_1$ for all t'' such that $t'' \in (t,t')$ and $(T,t'') \models_\Pi \top$;
- $(T,t) \models_\Pi \psi_1 \widetilde{\mathbf{U}}_I \psi_2$ iff for all $t' > t$ such that $t' - t \in I$ and $(T,t') \models_\Pi \top$, either $(T,t') \models_\Pi \psi_2$ or $(T,t'') \models_\Pi \psi_1$ for some t'' such that $t'' \in (t,t')$ and $(T,t'') \models_\Pi \top$;
- $(T,t) \models_\Pi \exists \pi\, \varphi$ iff there is a trace $\rho \in T$ such that $(T,t) \models_{\Pi[\pi \mapsto \rho]} \varphi$;
- $(T,t) \models_\Pi \forall \pi\, \varphi$ iff for all traces $\rho \in T$, $(T,t) \models_{\Pi[\pi \mapsto \rho]} \varphi$.

We say that T *satisfies* a closed HyperMTL formula φ in the asynchronous semantics ($T \models \varphi$) iff $(T,0) \models_{\Pi_\emptyset} \varphi$.

The asynchronous semantics for HyperMTL is (arguably) the most natural choice of semantics for the current event-based setting. As the examples below illustrate, allowing explicit interleaving of events may simplify the specification even when no quantitative timing constraint is involved.

▶ **Example 3.** Consider again the system in Example 2 and a low-security user u_L who can observe $\{B^0, B^1, C^0, C^1\}$ but not $\{A^0, A^1\}$. The property "if B^0 occurs in both π_a and π_b, then the corresponding C^0's must occur simultaneously in both π_a and π_b" (a variant of noninference [46]) can be specified with the following HyperMTL formula in the asynchronous semantics:

$$\varphi_1 = \forall \pi_a \forall \pi_b \left(\mathbf{F}\, B_a^0 \wedge \mathbf{F}\, B_b^0 \Rightarrow \mathbf{F}(C_a^0 \wedge C_b^0) \right).$$

In particular, $C_a^0 \wedge C_b^0$ holds only when the two $\{C^0\}$-events occur simultaneously in π_a and π_b. It is clear that the system does not satisfy φ_1, as there are two traces of the system where B^0 occurs in both, but the occurrences of C^0 are at different times; as we mentioned earlier, this allows u_L to infer A by timing C^0. If, on the other hand, the timing accuracy attainable by u_L is limited and thus it can only differentiate events that are d time units apart, the system can instead be checked against

$$\varphi_2 = \forall \pi_a \forall \pi_b \left(\mathbf{F}\, B_a^0 \wedge \mathbf{F}\, B_b^0 \Rightarrow \mathbf{F} \left(C_a^0 \wedge (\mathbf{F}_{\leq d}\, C_b^0 \vee \mathbf{O}_{\leq d}\, C_b^0) \right) \right)$$

where $\mathbf{O}$ is the past version of $\mathbf{F}$. This will be satisfied if $T_B - T_A \leq d$, and since u_L will not be able to infer A, the system may be considered secure in this case. Finally, note that in the original (synchronous) semantics for HyperLTL [16], φ_1 is satisfied by the system, as events are synchronised by their positions rather than times of occurrence.

▶ **Example 4** (Noninterference in event-based systems [31])**.** A system operating on sequences of commands issued by different users can be modelled as a deterministic finite automaton $\mathcal{A}$ over $\Sigma = U \times C$ where U is the set of users and C is the set of commands. Additionally, let Obs be the set of observations and $out : S \times U \to \mathsf{Obs}$ be the observation function for what can be observed at each location by each user. Let there be a partition of U into two disjoint sets of users $U_H \subseteq U$ and $U_L \subseteq U$. *Noninterference* requires that for each $w \in \Sigma^*$ where w ends with a command issued by a user in U_L and $\mathcal{A}$ reaches s after reading w, the subsequence w' obtained by removing all the commands issued by the users in U_H results in a location s' such that the observation $out(s', u_L)$ of each user $u_L \in U_L$ is identical to $out(s, u_L)$. For our purpose, we can combine $\mathcal{A}$ and out (in the expected way) into an automaton $\mathcal{A}'$ over Σ_{AP} where $\mathsf{AP} = (U \times C) \uplus (U \times \mathsf{Obs})$ (atomic propositions in $U \times \mathsf{Obs}$ reflect the observations at the location that has just been entered). Checking noninterference then amounts to model checking $\mathcal{A}'$ (whose locations are all accepting) against the following HyperMTL formula in the asynchronous semantics:

$$\varphi_3 = \forall \pi_a \forall \pi_b \big(\mathbf{G}(\top_b \Rightarrow \psi_b^L \wedge \psi_{U,C}^{=}) \wedge \mathbf{G}(\top_a \wedge \bot_b \Rightarrow \psi_a^H) \Rightarrow \mathbf{G}(\top_b \Rightarrow \psi_{out(U_L)}^{=}) \big)$$

(where ψ_b^L asserts that the command in π_b is issued by a user in U_L, $\psi_{U,C}^{=}$ says that the two synchronised commands in π_a and π_b agree on U and C, etc.). Specifically,

- $\mathbf{G}(\top_b \Rightarrow \psi_b^L \wedge \psi_{U,C}^{=})$ asserts that π_b only contains low commands and π_a also contains these commands at the exactly same times;
- $\mathbf{G}(\top_a \wedge \bot_b \Rightarrow \psi_a^H)$ asserts that all the commands that are only present in π_a are high commands;
- $\mathbf{G}(\top_b \Rightarrow \psi_{out(U_L)}^{=})$ ensures that, after each low command in π_b, the observation of each $u_L \in U_L$ is identical to the observation of u_L after the corresponding low command in π_a, regardless of the high commands that occur in the preceding 'gaps'.

We remark that while this example is essentially untimed, the asynchronous event-based formulation leads to a much simpler and clearer specification than the state-based one in [16].

Synchronous semantics. A less general semantics can be defined for HyperMTL formulae where each trace quantifier only ranges over traces that synchronise with the traces in the current trace assignment (this is the case in the original HyperLTL semantics [16]). For example, the second quantifier in $\exists \pi_a \, \exists \pi_b \, \psi$ requires π_b to satisfy $(\pi_a, t) \models \top_a \Leftrightarrow (\pi_b, t) \models \top_b$ for all $t \in \mathbb{R}_{\geq 0}$. The HyperMTL *synchronous* satisfaction relation $\models^{sync}$ can, in fact, be expressed in the asynchronous semantics by explicitly requiring newly quantified traces to synchronise in the quantifier-free part of the formula. More precisely, for a closed HyperMTL formula $\varphi = \mathcal{Q} \, \varphi'$ where $\mathcal{Q}$ denotes a block of quantifiers of the same type (i.e. all existential or all universal) and φ' is a possibly open HyperMTL formula, and a set V of trace variables, let (abusing notation slightly) $sync(\varphi, V) = \mathcal{Q} \left(\mathbf{G}(\bigwedge_{\pi \in \mathcal{Q} \cup V} \top_\pi) \wedge sync(\varphi', \mathcal{Q} \cup V) \right)$ when $\mathcal{Q}$ are existential, $sync(\varphi) = \mathcal{Q} \left(\mathbf{G}(\bigwedge_{\pi \in \mathcal{Q} \cup V} \top_\pi) \Rightarrow sync(\varphi', \mathcal{Q} \cup V) \right)$ when $\mathcal{Q}$ are universal, and $sync(\psi, V) = \psi$ when ψ is quantifier-free. The following lemma holds subject to rewriting the formula into prenex normal form.

▶ **Lemma 5.** *For any trace set T over Σ_{AP} and closed* HyperMTL *formula φ over* AP*, $T \models^{sync} \varphi$ iff $T \models sync(\varphi, \emptyset)$.*

While the synchronous semantics may seem quite restricted (intuitively, the chance that two random traces of a timed system have exactly the same timestamps is certainly slim!), one can argue that it already suffices for many applications if *stuttering steps* are allowed. We will see later that for alternation-free HyperMTL, the asynchronous semantics can be emulated in the synchronous semantics using a 'weak inverse' of Lemma 5.

Satisfiability and model checking. Given a closed HyperMTL formula φ over AP, the *satisfiability* problem asks whether there is a *non-empty* trace set $T \subseteq T\Sigma_{\mathsf{AP}}^*$ satisfying it, i.e. $T \models \varphi$ (or $T \models^{sync} \varphi$, if the synchronous semantics is assumed). Given a TA $\mathcal{A}$ over Σ_{AP} and a closed HyperMTL formula φ over AP, the *model-checking* problem asks whether $[\![\mathcal{A}]\!] \models \varphi$ (or $[\![\mathcal{A}]\!] \models^{sync} \varphi$). Our focus in this paper is on the *decidability* of these problems, as their complexity (when they are decidable) follow straightforwardly from standard results on MTL [5] and HyperLTL [16, 22].

3 Satisfiability

To emulate interleaving of events (of a concurrent or distributed system, say) in a synchronous, state-based setting, it is natural and necessary to introduce stuttering steps. In the context of verification, it is often a desirable trait for a temporal logic to be *stutter-invariant* [41, 42] so that it cannot be used to differentiate traces that ought to be regarded as the same (e.g., in an iterative refinement process, an abstract component of a system

may be replaced by a concrete implementation that simulates an abstract step with some additional internal actions). As a simple attempt to reconcile the asynchronous and synchronous semantics of HyperMTL, we can make use of *silent events* in the same spirit to enable synchronisation of interleaving traces while preserving the semantics. More precisely, let $\mathit{stutter}(\rho)$ for a trace $\rho \in T\Sigma^*_{\mathsf{AP}}$ be the maximal set of traces $\rho' \in T\Sigma^*_{\mathsf{AP}_\epsilon}$ ($\mathsf{AP}_\epsilon = \mathsf{AP} \cup \{p^\epsilon\}$) such that

- for every event (σ_i, τ_i) in ρ', either $\sigma_i = \{p^\epsilon\}$ or $p^\epsilon \notin \sigma_i$;
- ρ can be obtained from ρ' by deleting all the $\{p^\epsilon\}$-events.

This extends to trace sets $T \subseteq T\Sigma^*_{\mathsf{AP}}$ in the obvious way. For a closed alternation-free HyperMTL formula $\varphi = \mathcal{Q}\psi$ over AP, let $\mathit{stutter}(\varphi) = \mathcal{Q}\psi''$ be the HyperMTL formula over AP_ϵ obtained by replacing in ψ, e.g., all $\top_\pi$ with $\neg p^\epsilon_\pi$, to give ψ', and finally let $\psi'' = \mathbf{G}(\bigvee_{\pi\in\mathcal{Q}} \neg p^\epsilon_\pi) \wedge \big(\bigwedge_{\pi\in\mathcal{Q}} \mathbf{G}(p^\epsilon_\pi \Rightarrow \bigwedge_{p\in\mathsf{AP}} \neg p_\pi)\big) \wedge \psi'$ when $\mathcal{Q}$ are existential and $\psi'' = \mathbf{G}(\bigvee_{\pi\in\mathcal{Q}} \neg p^\epsilon_\pi) \wedge \big(\bigwedge_{\pi\in\mathcal{Q}} \mathbf{G}(p^\epsilon_\pi \Rightarrow \bigwedge_{p\in\mathsf{AP}} \neg p_\pi)\big) \Rightarrow \psi'$ when $\mathcal{Q}$ are universal. Intuitively, ψ'' ensures that the traces involved are *well-formed* (i.e. satisfy the first condition above), and its own satisfaction is insensitive to the addition of silent events. The following lemma follows from a simple structural induction.

▶ **Lemma 6.** *For any trace set T over Σ_{AP} and closed alternation-free* HyperMTL *formula $\varphi = \mathcal{Q}\psi$ over* AP *($\mathcal{Q}$ is either a block of existential quantifiers or universal quantifiers and ψ is quantifier-free), $T \models \varphi$ iff $\mathit{stutter}(T) \models^{sync} \mathit{stutter}(\varphi)$.*

The following two lemmas follow from Lemma 6 and the fact that for alternation-free HyperMTL formulae, satisfiability in the synchronous semantics can be reduced (in the same way as HyperLTL) to MTL satisfiability.

▶ **Lemma 7.** *The satisfiability problem for $\exists^*$-*HyperMTL *is decidable.*

▶ **Lemma 8.** *The satisfiability problem for $\forall^*$-*HyperMTL *is decidable.*

Lemma 6, however, does not extend to larger fragments of HyperMTL. For example, consider $T = \{(\{p\},1)(\{r\},3),(\{q\},2)\}$ and $\varphi = \exists\pi_a\,\forall\pi_b\,(\mathbf{F}\,p_a \wedge \neg\,\mathbf{F}\,q_b)$. Now it is obvious that $T \not\models \varphi$, but since $(\{p\},1)(\{r\},3) \in \mathit{stutter}(T)$, we have $\mathit{stutter}(T) \models^{sync} \mathit{stutter}(\varphi)$ (provided that the definition of $\mathit{stutter}(\cdot)$ is extended to general HyperMTL formulae, as in Lemma 5). Still, it is not hard to see that the crucial observation used in $\exists^*\forall^*$-HyperLTL satisfiability (if $\exists\pi_0 \ldots \exists\pi_k\,\forall\pi'_0 \ldots \forall\pi'_\ell\,\psi$ is satisfiable, then it is also satisfiable by the trace set $\{\pi_0, \ldots \pi_k\}$) extends to HyperMTL in the asynchronous semantics; the following lemma then follows from Lemma 7.

▶ **Lemma 9.** *The satisfiability problem for $\exists^*\forall^*$-*HyperMTL *is decidable.*

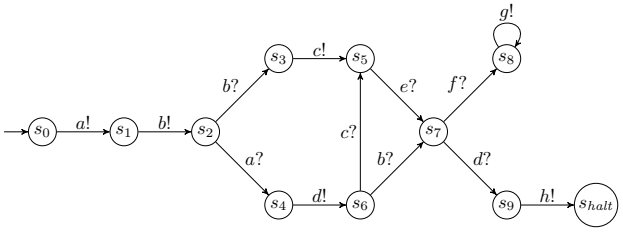
Finally, note that the undecidability of $\forall\exists$-HyperLTL carries over to HyperMTL: in the synchronous semantics, the reduction in [22] applies directly with some trivial modifications (as we work with finite traces); undecidability then holds for the case of asynchronous semantics as well, by Lemma 5.

▶ **Lemma 10.** *The satisfiability problem for $\forall\exists$-*HyperMTL *is undecidable.*

▶ **Theorem 11.** *The satisfiability problem for* HyperMTL *is decidable if the formula does not contain $\forall\exists$.*

4 Model checking

We now turn to the model-checking problem, which behaves quite differently than in the case of HyperLTL.



$$(s_0, \epsilon) \to (s_1, a) \to (s_2, ab) \to (s_4, b) \to (s_6, bd) \to (s_7, d) \to (s_9, \epsilon) \to (s_{halt}, h)$$

Figure 4 A DCM and its unique halting computation.

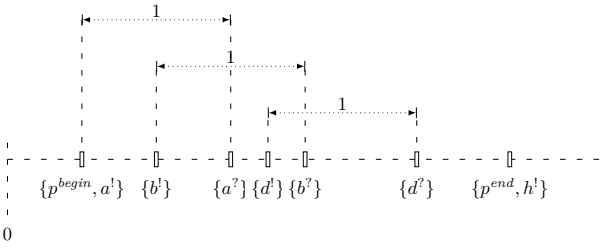


Figure 5 A trace encoding the halting computation of the DCM in Figure 4. Note that each $m^!$ is followed by a corresponding $m^?$ exactly 1 time unit later.

The alternation-free case. Without loss of generality, we consider only the case of $\exists^*$-HyperMTL in the asynchronous semantics. By Lemma 6, checking $[\![\mathcal{A}]\!] \models \varphi$ (for a TA $\mathcal{A}$ over Σ_{AP} and a closed $\exists^*$-HyperMTL formula φ over AP) is equivalent to checking $\mathit{stutter}([\![\mathcal{A}]\!]) \models^{sync} \mathit{stutter}(\varphi)$. To this end, we define $\mathit{stutter}(\mathcal{A})$ as the TA over $\Sigma_{\mathsf{AP}_\epsilon}$ obtained from $\mathcal{A}$ by adding a self-loop labelled with $\{p^\epsilon\}$ to each location; it should be clear that $[\![\mathit{stutter}(\mathcal{A})]\!] = \mathit{stutter}([\![\mathcal{A}]\!])$. In this way, the problem reduces to model checking $\exists^*$-HyperMTL in the synchronous semantics which, as the model-checking problem for $\exists^*$-HyperLTL, can be reduced to MTL model checking.

▶ **Theorem 12.** *Model checking alternation-free* HyperMTL *is decidable.*

The general case. Recall that the model-checking problem for HyperLTL is decidable even when the specification involves arbitrary nesting of quantifiers. This is unfortunately not the case for HyperMTL: allowing only one quantifier alternation already leads to undecidability. To see this, recall that any TA can be written as a formula $\exists X\, \psi$ where X is a set of (new) atomic propositions and ψ is an MTL formula [33, 53]. The undecidable TA universality problem—given a TA $\mathcal{A}$ over Σ, deciding whether $[\![\mathcal{A}]\!] = T\Sigma^*$—can thus be reduced to model checking HyperMTL: one simply checks whether there exists an X-labelling for every timed word over Σ so that ψ is satisfied. Here we show that model checking HyperMTL is essentially a harder problem: in the case of asynchronous semantics, model checking HyperMTL with quantifier alternations necessarily involves TAs with ϵ-transitions [11], and therefore remains undecidable even when *both the model and the specification are deterministic and only one of them uses a single clock (i.e. the other is untimed)*; by contrast, (standard) TA universality over finite timed words is decidable when the TA uses only one clock [49].

We adapt the undecidability proof of the *reactive synthesis* problem for MTL in [14], which itself is by reduction from the halting problem for *deterministic channel machines* (DCMs), known to be undecidable [13]. Note that, in contrast to HyperMTL model checking,

MTL reactive synthesis is decidable when the specification is deterministic [19]; in this sense, quantification over traces is more powerful than quantification over strategies (there is a winning strategy of the controller for all possible strategies of the environment).[6] For our purpose, we introduce the $\triangleleft_I$ operator, in which we allow I to be singular (note that this is merely syntactic sugar and does not increase the expressiveness of MTL [33, 53]):

- $(T,t) \models_\Pi \triangleleft_I \varphi$ iff there exists t', $0 \leq t' < t$ such that $t - t' \in I$, $(T,t') \models_\Pi \top$, $(T,t') \models_\Pi \varphi$, and $(T,t'') \not\models_\Pi \varphi$ for all t'' such that $t'' \in (t',t)$ and $(T,t'') \models_\Pi \top$.

Let $\mathsf{LTL}_\triangleleft$ be the fragment of MTL where all timed subformulae must be of the form $\triangleleft_I \varphi$, and all φ's in such subformulae must be 'pure past' formulae; these requirements ensure that $\mathsf{LTL}_\triangleleft$, in which we will write the quantifier-free part of the specification, translates into deterministic TAs [18]. To ease the understanding, we will first do the proof for the case of asynchronous semantics and then adapt it to the case of synchronous semantics.

▶ **Theorem 13.** *Model checking* $\exists^*\forall^*$-HyperMTL *and* $\forall^*\exists^*$-HyperMTL *are undecidable in the asynchronous semantics.*

Proof. A DCM $\mathcal{S} = \langle S, s_0, s_{halt}, M, \Delta \rangle$ can be seen as a finite automaton equipped with an unbounded fifo channel: S is a finite set of locations, s_0 is the initial location, s_{halt} is the halting location (such that $s_{halt} \neq s_0$), M is a finite set of messages, and $\Delta \subseteq S \times \{m!, m? \mid m \in M\} \times S$ is the transition relation satisfying the following determinism hypothesis: (i) $(s,q,s') \in \Delta$ and $(s,q,s'') \in \Delta$ implies $s' = s''$; (ii) if $(s,m!,s') \in \Delta$ then it is the only outgoing transition from s. Without loss of generality, we further assume that there is no incoming transition to s_0, no outgoing transition from s_{halt}, and $(s_0,q,s') \in \Delta$ implies that $q \in \{m! \mid m \in M\}$ and $s' \neq s_{halt}$. The semantics of $\mathcal{S}$ can be described with a graph $G(\mathcal{S})$ with vertices $\{(s,x) \mid s \in S, x \in M^*\}$ and edges defined as follows: (i) $(s,x) \to (s', xm)$ if $(s,m!,s') \in \Delta$; (ii) $(s,mx) \to (s',x)$ if $(s,m?,s') \in \Delta$. In other words, $m!$ 'writes' a copy of m to the channel and $m?$ 'reads' a copy of m off the channel. We say that $\mathcal{S}$ *halts* if there is a path in $G(\mathcal{S})$ from (s_0,ϵ) to (s_{halt},x) (a *halting computation* of $\mathcal{S}$) for some $x \in M^*$. An example DCM and its unique halting computation are depicted in Figure 4.

The idea, as in many similar proofs (e.g., [50]), is to encode a halting computation of $\mathcal{S}$ as a trace where each $m?$ is preceded by a corresponding $m!$ exactly 1 time unit earlier, and each $m!$ is followed by an $m?$ exactly 1 time unit later if s_{halt} has not been reached yet. To this end, let the model $\mathcal{A}$ be an (untimed) finite automaton over $\Sigma = 2^{\mathsf{AP}}$ where $\mathsf{AP} = \{m^!, m^? \mid m \in M\} \cup \{p^{begin}, p^{end}, p^{read}, p^1, q^1\}$ and whose set of locations is $S \cup \{s_1\}$, where s_1 is a new non-accepting location. The transitions of $\mathcal{A}$ follow $\mathcal{S}$: for each $m \in M$, $s \xrightarrow{\{m^?\}} s'$ is a transition of $\mathcal{A}$ iff $(s,m?,s') \in \Delta$, and similarly for $m!$—except for those going out of s_0 or going into s_{halt}, on which we further require p^{begin} or p^{end} to hold, respectively. Let s_0 be the initial location and s_{halt} be the only accepting location, and finally add transitions $s_0 \xrightarrow{\{p^{read}\}} s_{halt}$ and $s_0 \xrightarrow{\{p^1\}} s_1 \xrightarrow{\{q^1\}} s_{halt}$. It is clear that $\mathcal{A}$ is deterministic and it accepts only three types of traces:

1. From s_0 through some other locations of $\mathcal{S}$ and finally s_{halt}, i.e. those respecting the transition relation, but not necessarily the semantics, of $\mathcal{S}$.
2. From s_0 to s_{halt} in a single transition (on which p^{read} holds).
3. From s_0 to s_1 and then s_{halt}.

It remains to write a specification φ such that $[\![\mathcal{A}]\!] \models \varphi$ exactly when $\mathcal{A}$ accepts a trace of type (1) that also respects the semantics of $\mathcal{S}$ (one such trace that corresponds to the unique

[6] Indeed, the quantifier-free part ψ in the simpler encoding mentioned above (based on labelling timed words with propositions in X) is already in $\mathsf{LTL}_\triangleleft$ and thus is deterministic.

halting computation of the DCM in Figure 4 is depicted in Figure 5). This is where the traces of types (2) and (3) come into play: for example, if a trace of type (1) issues a read $m?$ without a corresponding write $m!$, then a trace of type (3) can be used to 'pinpoint' the error. More precisely, let $\varphi = \exists \pi_a \, \forall \pi_b \, (\psi_1 \wedge \psi_2 \wedge \psi_3 \wedge \psi_4)$ where

- $\psi_1 = \mathbf{F}\, p_a^{end}$ ensures that π_a is of type (1);
- $\psi_2 = \mathbf{F}(p_b^{read} \wedge \psi_R) \Rightarrow \mathbf{F}(p_b^{read} \wedge \lhd_{\geq 1} p_a^{begin})$, where $\psi_R = \bigvee\{m_a^? \mid m \in M\}$, is a simple sanity check which ensures that in π_a, each $m^?$ must happen at time $\geq t+1$ if p^{begin} happens at t;
- $\psi_3 = \bigwedge_{m \in M} \Big(\mathbf{F}(q_b^1 \wedge m_a^?) \Rightarrow \big(\mathbf{F}(p_a^{begin} \wedge \mathbf{F}\, p_b^1) \wedge \mathbf{F}(q_b^1 \wedge \lhd_{=1} p_b^1) \Rightarrow \mathbf{F}(p_b^1 \wedge m_a^!) \big) \Big)$ ensures that each $m^?$, if it happens at t, is preceded by a corresponding $m^!$ at $t-1$ in π_a;
- $\psi_4 = \bigwedge_{m \in M} \Big(\mathbf{F}(p_b^1 \wedge m_a^!) \Rightarrow \mathbf{F}(p_a^{end} \wedge \lhd_{<1} p_b^1) \vee \big(\mathbf{F}(q_b^1 \wedge \lhd_{=1} p_b^1) \Rightarrow \mathbf{F}(q_b^1 \wedge m_a^?) \big) \Big)$ ensures that each $m^!$ at t is followed by a corresponding $m^?$ at $t+1$ (unless p^{end} happens first) in π_a.

Now observe that the only timed subformulae are $\lhd_{\geq 1} p_a^{begin}$, $\lhd_{=1} p_b^1$, and $\lhd_{<1} p_b^1$. As p^1 and p^{read} cannot happen in the same trace (π_b), it is not hard to see that the reduction remains correct if we replace these by $\lhd_{\geq 1}(p_a^{begin} \vee p_b^1)$, $\lhd_{=1}(p_a^{begin} \vee p_b^1)$, and $\lhd_{<1}(p_a^{begin} \vee p_b^1)$ (respectively) to obtain ψ_2', ψ_3', and ψ_4'. It follows that $\psi_1 \wedge \psi_2' \wedge \psi_3' \wedge \psi_4'$ can be translated into a one-clock deterministic TA. Finally, it is possible to move all the timing constraints into the model and use an untimed HyperLTL formula as the specification: in the model, ensure that p^1 and q^1 are separated by exactly 1 time unit, and add $s_0 \xrightarrow{\{p^2\}} s_1 \xrightarrow{\{q^2\}} s_{halt}$ such that p^2 and q^2 are separated by < 1 time unit; in the specification, use p^2, q^2 to rule out those π_a's with some $m^?$ at < 1 time unit from p^{begin}. ◀

Now we consider the synchronous semantics. The corresponding result is weaker in this case, as we will see in the next section that in several subcases the problem becomes decidable. Still, the reduction above can be made to work if the model has one clock and an extra trace quantifier is allowed.

▶ **Theorem 14.** *Model checking* $\exists^*\forall^*$-HyperMTL *and* $\forall^*\exists^*$-HyperMTL *are undecidable in the synchronous semantics.*

Proof of Theorem 14. We use a modified model $\mathcal{A}'$ whose set of locations is $S \cup \{s_1, s_2, s_3, s_4\}$; the transitions are similar to $\mathcal{A}$ in the proof of Theorem 13, but we now use a clock x in the path $s_0 \xrightarrow[x:=0]{\{p^1\}} s_1 \xrightarrow[x \geq 1, x:=0]{\{q^1\}} s_{halt}$, the paths $s_0 \xrightarrow[x:=0]{\{p^2\}} s_2 \xrightarrow[x \leq 1, x:=0]{\{q^2\}} s_{halt}$, $s_0 \xrightarrow[x:=0]{\{p^3\}} s_3 \xrightarrow[x > 1, x:=0]{\{q^3\}} s_{halt}$, $s_0 \xrightarrow[x:=0]{\{p^4\}} s_4 \xrightarrow[x < 1, x:=0]{\{q^4\}} s_{halt}$ are added, and $s_0 \xrightarrow{\{p^{read}\}} s_{halt}$ is removed. Moreover, a self-loop labelled with $\{p^\epsilon\}$ is added to each of s_0, s_1, s_2, s_3, s_4, and s_{halt}. The specification is $\varphi' = \exists \pi_a \, \forall \pi_b \, \forall \pi_c \bigwedge_{1 \leq i \leq 9} \psi_i'$ where $\bigwedge_{1 \leq i \leq 9} \psi_i'$ is the following untimed LTL formula:

- $\psi_1' = \mathbf{F}\, p_a^{end}$;
- $\psi_2' = \mathbf{F}(q_b^4 \wedge \psi_R) \Rightarrow \neg\, \mathbf{F}(p_b^4 \wedge p_a^{begin})$ where $\psi_R = \bigvee\{m_a^? \mid m \in M\}$;
- $\psi_3' = \bigwedge_{m \in M} \big(\mathbf{F}(q_b^1 \wedge q_c^2 \wedge m_a^?) \wedge \mathbf{F}(p_b^1 \wedge p_c^2) \Rightarrow \mathbf{F}(p_b^1 \wedge p_c^2 \wedge m_a^!) \big)$;
- $\psi_4' = \mathbf{F}(q_b^3 \wedge \psi_R) \Rightarrow \neg\, \mathbf{F}(p_b^3 \wedge \mathbf{X}\, q_b^3)$;
- $\psi_5' = \mathbf{F}(q_b^3 \wedge q_c^4 \wedge \psi_R) \Rightarrow \neg\, \mathbf{F}(p_b^3 \wedge \mathbf{X}\, p_c^4)$;
- $\psi_6' = \mathbf{F}(p_b^4 \wedge \psi_W) \Rightarrow \neg\, \mathbf{F}(q_b^4 \wedge \neg\, \mathbf{X}\, \top)$ where $\psi_W = \bigvee\{m_a^! \mid m \in M\}$;
- $\psi_7' = \bigwedge_{m \in M} \Big(\mathbf{F}(p_b^1 \wedge p_c^2 \wedge m_a^!) \wedge \mathbf{F}(q_b^1 \wedge q_c^2) \Rightarrow \mathbf{F}\, \big(q_b^1 \wedge q_c^2 \wedge (m_a^? \vee p_a^\epsilon) \big) \Big)$;
- $\psi_8' = \mathbf{F}(p_b^3 \wedge \psi_W) \Rightarrow \neg\, \mathbf{F}(p_b^3 \wedge \mathbf{X}\, q_b^3)$;
- $\psi_9' = \mathbf{F}(p_b^3 \wedge p_c^4 \wedge \psi_W) \Rightarrow \neg\, \mathbf{F}(q_b^3 \wedge \mathbf{X}\, q_c^4)$.

In this modified reduction, ψ'_1, ψ'_2 play similar roles as ψ_1, ψ_2 in the proof of Theorem 13. ψ'_3 ensures that if each $m^?$ at t is preceded by an event at $t-1$, then $m^!$ must hold there. ψ'_4 and ψ'_5 ensures that each $m^?$ at t is actually preceded by an event at $t-1$. The roles of ψ'_6, ψ'_7, ψ'_8, and ψ'_8 are analogous (note the use of silent events at the end of π_a). ◀

Restricted models. We conclude this section by showing that the undecidability results above can actually be obtained for trivial systems with only a single location. In particular, the structural restrictions considered in [12] have no effect on the decidability of HyperMTL model checking.

▶ **Corollary 15.** *Model checking* $\exists^*\forall^*$-HyperMTL *and* $\forall^*\exists^*$-HyperMTL *are undecidable in the asynchronous semantics for systems with only one location.*

▶ **Corollary 16.** *Model checking* $\exists^*\forall^*$-HyperMTL *and* $\forall^*\exists^*$-HyperMTL *are undecidable in the synchronous semantics for systems with only one location.*

5 Decidable subcases

While the negative results in the previous section may be disappointing, we stress again that model checking alternation-free HyperMTL is no harder than MTL model checking, and it can in fact be carried out with algorithms and tools for the latter. In any case, we now identify several subcases where model checking is decidable beyond the alternation-free fragment.

Untimed model + untimed specification. The first case we consider is when both the model and the specification are untimed, and the asynchronous semantics is assumed (note that, if instead, the synchronous semantics is assumed, then this case is simply HyperLTL model checking). Our algorithm follows the lines of [16] and is essentially based on *self-composition* (cf. [10], and many others; see the references in [16]) of the model; the difficulty here, however, is to handle interleaving of events. Let the model $\mathcal{A}$ be a finite automaton over Σ_{AP} and the specification be a (untimed) closed HyperMTL formula over AP. Without loss of generality, we assume the specification to be $\varphi = \exists\pi_1\,\forall\pi_2\,\ldots\exists\pi_{k-1}\,\forall\pi_k\,\psi$, which can be rewritten into $\exists\pi_1\,\neg\exists\pi_2\,\neg\ldots\exists\pi_{k-1}\,\neg\exists\pi_k\,\neg\psi$. We start by translating $\mathit{stutter}(\neg\psi)$ (in which we replace all occurrences of $\top_i$ with $\neg p_i^\epsilon$, i.e. regarded here simply as an MTL formula over $(\mathsf{AP}_\epsilon)^k = \{p_i \mid p \in \mathsf{AP}_\epsilon, 1 \le i \le k\}$) into the equivalent finite automaton over $\Sigma_{(\mathsf{AP}_\epsilon)^k}$, and take its product with (i) the automaton for $\mathbf{G}(\bigvee_{1\le i\le k}\neg p_i^\epsilon) \wedge \big(\bigwedge_{1\le i\le k}\mathbf{G}(p_i^\epsilon \Rightarrow \bigwedge_{p\in\mathsf{AP}}\neg p_i)\big)$ and (ii) the automaton obtained from $\mathit{stutter}(\mathcal{A})$ by extending the alphabet to $\Sigma_{(\mathsf{AP}_\epsilon)^k}$ and renaming all the occurrences of p to p_k, to obtain $\mathcal{B}$. Now let $\mathcal{C}$ be the projection of $\mathcal{B}$ onto $(\mathsf{AP}_\epsilon)^{k-1} = \{p_i \mid p \in \mathsf{AP}_\epsilon, 1 \le i \le k-1\}$ (this step corresponds to $\exists$ in $\neg\exists\pi_k$). By construction, $\mathcal{B}$ accepts only traces that are well-formed in dimensions 1 to $k-1$, and so does $\mathcal{C}$; but $\mathcal{C}$ may accept traces containing $\{p_i^\epsilon \mid 1 \le i \le k-1\}$-events. We replace these events by ϵ (the 'real' silent event, which can be removed with the standard textbook constructions, e.g., [36]) to obtain $\mathcal{C}'$. Finally, we complement $\mathcal{C}'$ to obtain $\mathcal{C}''$ (this step corresponds to $\neg$ in $\neg\exists\pi_k$). We can then start over by taking the product of $\mathcal{C}''$, the automaton for $\mathbf{G}(\bigvee_{1\le i\le k-1}\neg p_i^\epsilon) \wedge \big(\bigwedge_{1\le i\le k-1}\mathbf{G}(p_i^\epsilon \Rightarrow \bigwedge_{p\in\mathsf{AP}}\neg p_i)\big)$, and the automaton obtained from $\mathit{stutter}(\mathcal{A})$ by extending the alphabet to $\Sigma_{(\mathsf{AP}_\epsilon)^{k-1}}$ and renaming all the occurrences of p to p_{k-1}; the resulting automaton is the new $\mathcal{B}$. We continue this process until the outermost quantifier $\exists\pi_1$ is reached, when we test the emptiness of $\mathcal{B}$ (at this point, it is an automaton over $\Sigma_{\mathsf{AP}_\epsilon}$).

▶ **Proposition 17.** *Model checking* HyperMTL *is decidable when the model and the specification are both untimed.*

One clock + one alternation. The algorithm outlined in the previous case crucially depends on the fact that both $\mathcal{A}$ and φ are untimed, hence their product (in the sense detailed in the previous case) can be complemented. When the synchronous semantics is assumed and there is only one quantifier alternation in φ, it might be the case that we do not actually need complementation. For example, if $\mathcal{A}$ is untimed and $\varphi = \forall \pi_a \, \exists \pi_b \, \exists \pi_c \, \psi$ where ψ translates into a one-clock TA, the corresponding model-checking problem clearly reduces to universality for one-clock TAs, which is decidable but non-primitive recursive [1].[7] This observation applies to other cases as well, such as when $\mathcal{A}$ is a one-clock TA and $\varphi = \exists \pi_a \, \forall \pi_b \, \psi$ where ψ is untimed; here model checking reduces to language inclusion between two one-clock TAs.

Untimed model + MIA specification. The main obstacle in applying the algorithm above to larger fragments of HyperMTL, as should be clear now, is that universal quantifiers amount to complementations, which are not possible in general in the case of TAs. Moreover, we note that the usual strategy of restricting to deterministic models and specifications does not help, as the projection step in the algorithm necessarily introduces non-determinism. To make the algorithm work for larger fragments, we essentially need a class of automata that is both *closed under projection* and *complementable*. Fortunately, there is a subclass of one-clock TAs that satisfies these conditions. We consider two additional restrictions on one-clock TAs:

- Non-Singular (NS): a one-clock TA is NS if all the guards are non-singular (i.e. must be of the form $x \in I$ where x is the single clock and I is a non-singular interval).
- Reset-on-Testing (RoT): a one-clock TA is RoT if whenever the guard of a transition is not $\top$, x must be reset on that transition.

One-clock TAs satisfying both NS and RoT are called *metric interval automata* (MIAs), which are determinisable [21]. Since the projection operation cannot invalidate NS and RoT, the algorithm above can be applied when the synchronous semantics is assumed, $\mathcal{A}$ is untimed, ψ or $\neg\psi$ translates to a MIA, and only one complementation is involved; in this case it runs in elementary time.

▶ **Proposition 18.** *Model checking* $\forall^*\exists^*$-HyperMTL *(*$\exists^*\forall^*$-HyperMTL*) is decidable in the synchronous semantics when the model is untimed and* ψ *(*$\neg\psi$*) translates into a* MIA *in the specification* $\varphi = \forall \pi_1 \ldots \exists \pi_k \, \psi$ *(*$\varphi = \exists \pi_1 \ldots \forall \pi_k \, \psi$*).*

On the other hand, we can adapt the proof of Theorem 14 to show that model checking an untimed model against an $\exists^*\forall^*$-HyperMTL specification φ in the synchronous semantics, when the quantifier-free part ψ (instead of $\neg\psi$) translates into a MIA, remains undecidable.

▶ **Proposition 19.** *Model checking* $\exists^*\forall^*$-HyperMTL *is undecidable in the synchronous semantics when the model is untimed and* ψ *in the specification* $\varphi = \exists \pi_1 \ldots \forall \pi_k \, \psi$ *translates into a* MIA*.*

The decidability results in the synchronous semantics are summarised in Table 1.

Bounded time domains. We end this section by showing that when there is an *a priori* bound N (where N is a positive integer) on the length of the time domain, the model-checking problem for full HyperMTL becomes decidable; in fact, in the case of synchronous

[7] This case is undecidable in the asynchronous semantics by Theorem 13; as explained above, the algorithm may introduce ϵ-transitions in the asynchronous semantics, while universality for one-clock TAs with ϵ-transitions is undecidable [1].

Model \ Spec.	untimed	NS+RoT	NS	RoT
untimed	Dec. (Proposition 17)	Dec. for $\forall^*\exists^*$ (Proposition 18)	Undec. for $\exists\forall\forall$ (Proposition 19)	Undec. for $\exists\forall\forall$ (Proposition 19)
NS+RoT	Undec. for $\exists\forall\forall$ (Theorem 14)	Undec.	Undec.	Undec.
NS	Undec.	Undec.	Undec.	Undec.
RoT	Undec.	Undec.	Undec.	Undec.

Table 1 Decidability of model checking untimed or one-clock TAs against (one-clock) HyperMTL in the synchronous semantics; NS stands for Non-Singular constraints and RoT stands for Reset-on-Testing.

semantics it reduces to the satisfiability problem for QPTL [56]. From a practical point of view, this implies that *time-bounded* HyperMTL verification (at least for the $\exists^*\forall^*$-fragment, say) can be carried out with highly efficient, off-the-shelf tools that work with LTL and (untimed) automata, such as SPOT [20], GOAL [57], and `Owl` [40].

We assume the asynchronous semantics. For a given N, we consider all traces in which all timestamps are less than N. Denote by $[\![\mathcal{A}]\!]_{[0,N)}$ the set of all such traces in $[\![\mathcal{A}]\!]$; the model-checking problem then becomes deciding whether $[\![\mathcal{A}]\!]_{[0,N)} \models \varphi$. As before, we assume φ to be $\exists\pi_1 \neg\exists\pi_2 \neg \ldots \exists\pi_{k-1} \neg\exists\pi_k \neg\psi$. Following [34, 48], we can use the *stacking construction* to obtain, from the conjunction ψ' of $\mathit{stutter}(\neg\psi)$ and $\mathbf{G}(\bigvee_{\pi\in\mathcal{Q}} \neg p^\epsilon_\pi) \wedge \big(\bigwedge_{\pi\in\mathcal{Q}} \mathbf{G}(p^\epsilon_\pi \Rightarrow \bigwedge_{p\in\mathsf{AP}} \neg p_\pi)\big)$, an equi-satisfiable untimed (QPTL) formula $\overline{\varphi} = \exists W\, \overline{\psi'}$ over the stacked alphabet $\overline{(\mathsf{AP}_\epsilon)^k} \cup \overline{Q}$ (where $\overline{(\mathsf{AP}_\epsilon)^k} = \{p_{i,j} \mid p \in \mathsf{AP}_\epsilon, 1 \le i \le k, 0 \le j < N\}$ and $\overline{Q} = \{q_j \mid 0 \le j < N\}$). We apply the following modifications to $\overline{\varphi}$ to obtain $\overline{\varphi'}$:

- introduce atomic propositions $\{p^\epsilon_i \mid 1 \le i \le k\}$ and add the conjunct

$$\wedge_{1\le i\le k} \mathbf{G}\left(\big(\wedge_{0\le j<N} (q_j \Rightarrow p^\epsilon_{i,j})\big) \Leftrightarrow p^\epsilon_i\right);$$

- introduce atomic propositions $\{q_{i,j} \mid 1 \le i \le k, 0 \le j < N\}$ and add the conjunct

$$\wedge_{1\le i\le k} \mathbf{G}\left(\wedge_{0\le j<N} (\neg p^\epsilon_i \wedge q_j \Leftrightarrow q_{i,j})\right);$$

- project away $\{p^\epsilon_{i,j} \mid 1 \le i \le k, 0 \le j < N\}$ and $\overline{Q}$;
- replace all occurrences of p^ϵ_i by $\bot_i$.

Now, as we mentioned earlier, we can write $\mathcal{A}$ as an (MSO[$<,+1$] [48]) formula $\varphi_\mathcal{A} = \exists X_\mathcal{A}\, \psi_\mathcal{A}$ where $X_\mathcal{A}$ is a set of atomic propositions such that $\mathsf{AP} \cap X_\mathcal{A} = \emptyset$ and $\psi_\mathcal{A}$ is an MTL formula over $\mathsf{AP} \cup X_\mathcal{A}$. Let $\overline{\varphi_\mathcal{A}}$ be its stacked counterpart $\exists \overline{X_\mathcal{A}}\, \exists Y\, \overline{\psi_\mathcal{A}}$; we translate $\overline{\varphi_\mathcal{A}}$ back into an untimed automaton $\overline{\mathcal{A}}$ over the stacked alphabet $\overline{\mathsf{AP}} \cup \overline{Q}$. The problem thus reduces to untimed model checking of $\overline{\mathcal{A}}$ against $\exists\pi_1\, \forall\pi_2\, \ldots \exists\pi_{k-1}\, \forall\pi_k\, \overline{\varphi'}$ in the asynchronous semantics, which is decidable by Proposition 17 ($\overline{\varphi'}$ has outermost existential propositional quantifiers, but clearly the equivalent automaton can be used directly in the algorithm).

Finally, note that the proof is simpler for the case of synchronous semantics: we can simply work with a (non-stuttering) MSO[$<,+1$] formula in all the intermediate steps without translating it into an automaton, and then check the satisfiability of the final formula by stacking it into a QPTL formula.

▶ **Proposition 20.** *Model checking* HyperMTL *is decidable when the time domain is* $[0, N)$*, where* N *is a given positive integer.*

References

1 Parosh Aziz Abdulla, Johann Deneux, Joël Ouaknine, Karin Quaas, and James Worrell. Universality analysis for one-clock timed automata. *Fundam. Inform*, 89(4):419–450, 2008.

2 Erika Ábrahám and Borzoo Bonakdarpour. HyperPCTL: A temporal logic for probabilistic hyperproperties. In *QEST*, volume 11024 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2018.

3 Shreya Agrawal and Borzoo Bonakdarpour. Runtime verification of k-safety hyperproperties in HyperLTL. In *CSF*, pages 239–252. IEEE Computer Society, 2016.

4 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

5 Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

6 Rajeev Alur and Thomas A. Henzinger. Back to the future: towards a theory of timed regular languages. In *FOCS*, pages 177–186. IEEE Computer Society, 1992.

7 Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In *REX*, volume 600 of *LNCS*, pages 74–106. Springer-Verlag, 1992.

8 Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993.

9 Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):164–169, 1994.

10 Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. *Mathematical Structures in Computer Science*, 21(6):1207–1252, 2011.

11 Béatrice Bérard, Antoine Petit, Volker Diekert, and Paul Gastin. Characterization of the expressive power of silent transitions in timed automata. *Fundam. Inform*, 36(2-3):145–182, 1998.

12 Borzoo Bonakdarpour and Bernd Finkbeiner. The complexity of monitoring hyperproperties. In *CSF*, pages 162–174. IEEE Computer Society, 2018.

13 D. Brand and P. Zafiropulo. On communicating finite state machines. *Journal of the ACM*, 30:323–342, 1983.

14 Thomas Brihaye, Morgane Estiévenart, Gilles Geeraerts, Hsi-Ming Ho, Benjamin Monmege, and Nathalie Sznajder. Real-time synthesis is hard! In *FORMATS*, volume 9884 of *LNCS*, pages 105–120. Springer, 2016.

15 Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV2: An opensource tool for symbolic model checking. In *CAV*, volume 2404 of *LNCS*, pages 359–364. Springer, 2002.

16 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *POST*, volume 8414 of *LNCS*, pages 265–284. Springer, 2014.

17 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

18 Laurent Doyen, Gilles Geeraerts, Jean-François Raskin, and Julien Reichert. Realizability of real-time logics. In *FORMATS*, volume 5813 of *LNCS*, pages 133–148. Springer, 2009.

19 Deepak D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In *STACS*, volume 2285 of *LNCS*, pages 571–582. Springer, 2002.

20 Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 - a framework for LTL and ω-automata manipulation. In *ATVA*, volume 9938 of *LNCS*, pages 122–129. Springer, 2016.

21 Thomas Ferrère. The compound interest in relaxing punctuality. In *FM*, volume 10951 of *LNCS*, pages 147–164. Springer, 2018.

22 Bernd Finkbeiner and Christopher Hahn. Deciding hyperproperties. In *CONCUR*, volume 59 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

23 Bernd Finkbeiner, Christopher Hahn, and Tobias Hans. Mghyper: Checking satisfiability of HyperLTL formulas beyond the $\exists^*\forall^*$ fragment. In *ATVA*, volume 11138 of *Lecture Notes in Computer Science*, pages 521–527. Springer, 2018.

24 Bernd Finkbeiner, Christopher Hahn, and Marvin Stenger. Eahyper: Satisfiability, implication, and equivalence checking of hyperproperties. In *CAV*, volume 10427 of *Lecture Notes in Computer Science*, pages 564–570. Springer, 2017.

25 Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. In *RV*, volume 10548 of *LNCS*, pages 190–207. Springer, 2017.

26 Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. RVHyper: A runtime verification tool for temporal hyperproperties. In *TACAS*, volume 10806 of *Lecture Notes in Computer Science*, pages 194–200. Springer, 2018.

27 Bernd Finkbeiner, Christopher Hahn, and Hazem Torfah. Model checking quantitative hyperproperties. In *CAV*, volume 10981 of *Lecture Notes in Computer Science*, pages 144–163. Springer, 2018.

28 Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and $HyperCTL^*$. In *CAV*, volume 9206 of *LNCS*, pages 30–48. Springer, 2015.

29 Guillaume Gardey, John Mullins, and Olivier H. Roux. Non-interference control synthesis for security timed automata. *Electr. Notes Theor. Comput. Sci*, 180(1):35–53, 2007.

30 Christopher Gerking, David Schubert, and Eric Bodden. Model checking the information flow security of real-time systems. In *ESSoS*, volume 10953 of *LNCS*, pages 27–43. Springer, 2018.

31 J. A. Goguen and J. Meseguer. Security policies and security models. In *S&P*, pages 11–20. IEEE Computer Society, 1982.

32 Jens Heinen. Model checking timed hyperproperties. Master's thesis, Saarland University, 2018.

33 Thomas A. Henzinger, Jean-François Raskin, and Pierre-Yves Schobbens. The regular real-time languages. In *ICALP*, volume 1443 of *LNCS*, pages 580–591. Springer, 1998.

34 Hsi-Ming Ho. On the expressiveness of metric temporal logic over bounded timed words. In *RP*, volume 8762 of *Lecture Notes in Computer Science*, pages 138–150. Springer, 2014.

35 Gerard J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

36 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

37 Marieke Huisman, Pratik Worah, and Kim Sunesen. A temporal logic characterisation of observational determinism. In *CSFW*, page 3. IEEE Computer Society, 2006.

38 Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *CoRR*, abs/1801.01203, 2018.

39 Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

40 Jan Kretínský, Tobias Meggendorfer, and Salomon Sickert. Owl: A library for ω-words, automata, and ltl. In *ATVA*, volume 11138 of *Lecture Notes in Computer Science*, pages 543–550. Springer, 2018.

41 Antonín Kučera and Jan Strejček. The stuttering principle revisited. *Acta Informatica*, 41(7–8):415–434, 2005.

42 L. Lamport. What good is temporal logic? In R.E.A. Mason, editor, *IFIP Congress*, pages 657–667, Amsterdam, 1983. North-Holland.

43 Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

44 Yang Li, Kazuo Sakiyama, Shigeto Gomisawa, Toshinori Fukunaga, Junko Takahashi, and Kazuo Ohta. Fault sensitivity analysis. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 320–334. Springer, 2010.

45 Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *USENIX Security Symposium*, pages 973–990. USENIX Association, 2018.

46 John McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *S&P*, pages 79–93. IEEE Computer Society, 1994.

47 Luan Viet Nguyen, James Kapinski, Xiaoqing Jin, Jyotirmoy V. Deshmukh, and Taylor T. Johnson. Hyperproperties of real-valued signals. In *MEMOCODE*, pages 104–113. ACM, 2017.

48 Joël Ouaknine, Alexander Rabinovich, and James Worrell. Time-bounded verification. In *Proceedings of CONCUR 2009*, volume 5710 of *LNCS*, pages 496–510. Springer, 2009.

49 Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *LICS*, pages 54–63. IEEE Computer Society, 2004.

50 Joël Ouaknine and James Worrell. On the decidability and complexity of metric temporal logic over finite words. *Logical Methods in Computer Science*, 3(1), 2007.

51 Joël Ouaknine and James Worrell. Some recent results in metric temporal logic. In *FORMATS*, volume 5215 of *LNCS*, pages 1–13. Springer, 2008.

52 Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.

53 Jean-François Raskin. *Logics, automata and classical theories for deciding real time*. PhD thesis, FUNDP (Belgium), 1999.

54 A. W. Roscoe. Csp and determinism in security modelling. In *S&P*, pages 114–127. IEEE Computer Society, 1995.

55 Laurent Simon, David Chisnall, and Ross J. Anderson. What you get is what you C: Controlling side effects in mainstream C compilers. In *EuroS&P*, pages 1–15. IEEE, 2018.

56 A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for Büchi automata with applications to temporal logic (extended abstract). In *ICALP*, volume 194 of *LNCS*, pages 465–474. Springer, 1985.

57 Ming-Hsien Tsai, Yih-Kuen Tsay, and Yu-Shiang Hwang. Goal for games, omega-automata, and logics. In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 883–889. Springer, 2013.

58 Panagiotis Vasilikos, Flemming Nielson, and Hanne Riis Nielson. Secure information release in timed automata. In *POST*, volume 10804 of *LNCS*, pages 28–52. Springer, 2018.

59 Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. In *CSFW*, page 29. IEEE Computer Society, 2003.