# Deciding branching hyperproperties for real time systems

Nabarun Deka*, Minjian Zhang*, Rohit Chadha†, and Mahesh Viswanathan*
*University of Illinois at Urbana-Champaign
{ndeka2, minjian2, vmahesh}@illinois.edu
†University of Missouri, Columbia
chadhar@missouri.edu

***Abstract*—Security properties of real-time systems often involve reasoning about hyper-properties, as opposed to properties of single executions or trees of executions. These hyper-properties need to additionally be expressive enough to reason about real-time constraints. Examples of such properties include information flow, side channel attacks and service-level agreements. In this paper we study computational problems related to a branching-time, hyper-property extension of metric temporal logic (MTL) that we call HCMTL*. We consider both the interval-based and point-based semantics of this logic. The verification problem that we consider is to determine if a given HCMTL* formula $\varphi$ is true in a system represented by a timed automaton. We show that this problem is undecidable. We then show that the verification problem is decidable if we consider executions upto a fixed time horizon $T$. Our decidability result relies on reducing the verification problem to the truth of an MSO formula over reals with a bounded time interval.**

## I. Introduction

Unlike the traditional safety and liveness properties, security guarantees such as non-interference desired of systems, are not *trace-based* [13], [25], and instead are properties of sets of executions. In their seminal paper, Clarkson and Schneider [13], called such requirements *hyperproperties*. Several temporal logics have been designed to express formally and reason about hyperproperties. By far, the most well-known of these logics are HyperLTL and HyperCTL* [12]. HyperLTL and HyperCTL* extend the standard temporal logics LTL [30] and CTL* [16] respectively. While HyperLTL allows for reasoning about linear time hyperproperties, HyperCTL* allows for reasoning about branching time hyperproperties [1]. HyperLTL and HyperCTL* differ from LTL and CTL* by having explicit path variables and thus allowing for quantification over multiple executing traces simultaneously. The problems of checking a finite-state system against HyperLTL and HyperCTL* formulas are shown to be decidable in [12] by showing that the verification problem reduces to the satisfiability problem for quantified propositional temporal logic QPTL [33]. QPTL is a generalization of LTL, and is interpreted over (untimed) transition systems.

While HyperLTL and HyperCTL* are able to express hyperproperties of transition systems, they are inadequate to express hyperproperties that relate "timed" executions, namely executions that are decorated by the *time* at which each observation of the system occurs. Such hyperproperties, henceforth referred to as timing hyperproperties, are essential to reason about the timing behaviors of a system. Some examples include the absence of timing leaks and timeliness of optimistic contract signing; see Section III for examples. The need for reasoning about timing hyperproperties has led to the development of timed hyperlogics, such as in [7], [9], [10], [23], [26], [31]. The real time system being analyzed in this context is usually modeled by *timed* automata [4] as in [9], [10], [23], [26]. In contrast, the logics in [7], [31] are geared towards verifying timed properties of cryptographic properties, and models are timed versions of the applied-pi calculus [2]. (See Section VI on Pager 13 for a detailed discussion of logics in [7], [31].)

A timed automaton is a finite-state automaton augmented with a finite set of clocks. The clocks progress synchronously, and the automaton can make transitions based on their values and reset any of its clocks during a transition. Verifying timed systems is more challenging, even for regular, non-hyperproperties. Thus, in [9], [10], the time model is taken to be discrete, *i.e.*, the timed traces are sequences of pairs of the observed state of the system and the time observed, where the times are non-negative integers. In contrast, [23] considers verifying timed hyperproperties of timed automata when the time model is taken to be continuous/dense. In particular, they consider the *point-based* semantics for timed automata [3], [6], [22]. The timed traces in the point-based semantics are also sequences of pairs of observed states and observed times, but unlike [9], [10], the observations may occur at times that are arbitrary non-negative real numbers. For specifying timing hyperproperties, [23] extends the linear-time logic Metric Interval Temporal Logic (MITL) [5] to Hyper Metric Interval Temporal Logic (HyperMITL) analogous to the extension of LTL to HyperLTL. MITL is a commonly used logic to specify properties of timed systems, and is similar to LTL except that the temporal modalities are annotated with non-singular[2] time intervals: for example, $\phi\, U_I\, \psi$ means that $\psi$ must be true at some time $t \in I$ units from the current time

[1] In linear time hyperproperties, the different executions being quantified are decided "in advance", and in branching time hyperproperties, an execution being quantified may "branch off" in the middle of the last quantified execution.

[2] A non-empty interval is singular if it has exactly one element. Otherwise, it is non-singular.

2024 IEEE 37th Computer Security Foundations Symposium (CSF) | 979-8-3503-6203-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/CSF61375.2024.00032



DOI 10.1109/CSF61375.2024.00032

and $\phi$ must hold at all times before $t$.

The problem of verifying timed automata against HyperMITL specifications is undecidable [23]. The same paper also establishes that the verification problem becomes decidable when restricting to executions all of whose observations occur before a given time-bound $N$. The decidability result is established by showing that the verification problem reduces to the satisfaction problem of QPTL. Time-bounded verification is an often-adapted strategy for taming the complexity of verifying timed systems [28], [29]. Please note that even though time of the observations may be bounded, the total number of observations within that time-bound is not bounded.

**Our contributions.** In this paper we consider the problem of verifying branching hyperproperties for timed automata. As has also been observed in [7], [31], several security guarantees for timed systems are branching hyperproperties; see Section III. In order to specify branching hyperproperties, we define the logic HCMTL* which is obtained by first extending the logic Metric Temporal Logic (MTL) [24]) to a branching-time logic (analogous to the extension of LTL to CTL*) and then considering the "hyper" version of the resulting logic (analogous to the extension of CTL* to HyperCTL*). MTL itself generalizes MITL by allowing singular intervals to annotate the temporal operators. The logic is presented in negation-normal form where negations are pushed down to propositions. Note that as a hyperproperty relates multiple executions, different timed traces in the hyperproperty may run for different times. This requires the logic to allow reasoning at times after the end of a trace. The negation normal form facilitates this when defining the semantics, as this allows for both a proposition and its negation to be false in a timed trace after it has ended. However, this means that we have to choose the temporal operators carefully to be able to express both the hyperproperty and its negation.

In a departure from [23], we also consider *interval-based* semantics [5], [21], [32] for timed automata in addition to point-based semantics. In the interval-based semantics, the system modeled by a timed automaton is continuously under observation, and the timed traces of the automaton are a sequence of pairs of the observed state and the interval during which the state is observed. Verifying timed automata with interval-based semantics is more difficult than with point-based semantics. For example, the problem of verifying timed automata against MTL specifications is undecidable for finite words in the interval-based semantics [22] but is decidable for finite words in the point-based semantics [27].

We consider two verification problems for both interval-based and point-based semantics.

**Bounded Time:** Given a timed automaton $\mathcal{A}$, an HCMTL* specification $\varphi$, and a time bound $N$, determine if $\varphi$ is satisfied by $\mathcal{A}$ when we consider only executions that are observed up-to time $N$.

**General:** Given a timed automaton $\mathcal{A}$ and an HCMTL* specification $\varphi$, determine if $\varphi$ is satisfied by $\mathcal{A}$.

**Bounded-Time verification.** Our first result is that the bounded-time verification is decidable for interval-based semantics. Note that as HyperMITL is a fragment of HCMTL*, this result generalizes the results of [23] to interval-based semantics. We make a few salient observations about the proof of this result.

Unlike [12], [23], it is not clear that the verification problem can be reduced to QPTL satisfaction for interval-based semantics. Instead, we choose to obtain decidability by showing the verification problem reduces to the problem of satisfaction of the Monadic Second Order Logic with strict inequality and successor (MSO$(<,+1)$), over the subset $[0,N)$ of reals. Specifically, we show that for each timed automaton $\mathcal{A}$ and HCMTL* formula $\varphi$, there is a MSO$(<,+1)$ formula $\psi_{\mathcal{A},\varphi}$ with a set of free monadic predicates $\mathsf{MP}_{\mathcal{A}}$ such that $\mathcal{A}$ satisfies $\varphi$ if and only if there is a model $f$ that satisfies $\psi_{\mathcal{A},\varphi}$. A model of the formula $\psi_{\mathcal{A},\varphi}$, hereafter referred to as a *flow*, is a function from the domain $[0,N)$ to the power-set of the set $\mathsf{MP}_{\mathcal{A}}$. Intuitively, for $t \in [0,N)$, $f(t)$ is the set of predicates $P \in \mathsf{MP}_{\mathcal{A}}$ that are true at $t$. In order to establish this result, we carefully construct a one-to-one mapping from the executions of $\mathcal{A}$ to the set of flows.

Our next result is that the bounded-time verification is decidable for point-based semantics, thus generalizing the results of [23] to branching hyperproperties for point-based semantics. The decidability result is established by showing that the bounded-time verification problem for point-based semantics is reducible to the bounded-time verification problem for interval-based semantics: for every timed automaton $\mathcal{A}_{\mathsf{pt}}$ and HCMTL* formula $\varphi_{\mathsf{pt}}$, there is a timed automaton $\mathcal{A}_{\mathsf{ib}}$ and a HCMTL* formula $\varphi_{\mathsf{ib}}$ constructible from $\mathcal{A}_{\mathsf{pt}}$ and $\varphi_{\mathsf{pt}}$ such that $\mathcal{A}_{\mathsf{ib}}$ satisfies $\varphi_{\mathsf{ib}}$ in the interval-based semantics if and only if $\mathcal{A}_{\mathsf{pt}}$ satisfies $\varphi_{\mathsf{pt}}$ in the point-based semantics. The key observation is that both timed automata and HCMTL* in the interval-based semantics are at least as expressive as the point-based semantics. In point-based semantics, we only consider time points where observations occur. Thus, we need to add a new proposition that marks these observations in the interval-based semantics. This reduction is valid even when the time domain is not bounded and for HyperMITL formulas.

**General verification.** Since verifying timed automata against MTL semantics is already undecidable [22] for interval-based semantics, the general verification problem for HCMTL* is undecidable for interval-based semantics. Further, as the reduction of the verification problem for point-based semantics to the verification problem for interval-based semantics is also valid for HyperMITL formulas, and verifying timed automata against HyperMITL specifications under point-based semantics is undecidable [23], the undecidability carries over to both point-based and interval-based semantics even for HyperMITL specifications.[3]

**Organization.** The rest of the paper is organized as follows.

[3]The undecidability proof in [23] uses past operators. We do not have past operators in our logic. Nevertheless, we can also show that the problem is undecidable by reducing the universality problem of timed automata to HyperMITL verification under point-based semantics.

The logic HCMTL* and its interval-based semantics for timed automata is presented in Section II. Section III discusses examples of timed hyperproperties and their formalization in HCMTL*. Section IV presents the decidability proof of bounded-time verification in the interval-based semantics, and Section V presents the reduction from point-based semantics to interval-based semantics. Related work is discussed in Section VI, and we present our conclusions in Section VII.

Due to space constraints, we are not able to present the proofs in their complete technical details in this paper. Instead, we present the key ideas of the proofs and illustrate them using examples. The complete proofs can be found in the extended version of the paper [14].

## II. The logic HCMTL*

In this section, we introduce our logic HCMTL* which allows one to express branching hyperproperties of real time systems. HCMTL* is an extension of *metric temporal logic* (MTL) [24] which is a linear time logic to reason about real time constraints. Our real time systems will be modeled by timed automata [4], which is a popular model to describe such systems. We begin by defining the syntax of our logic, introduce timed automata, and conclude by formally defining what it means for a timed automaton to satisfy a formula in HCMTL*. As is standard, we use $\mathbb{N}$ for natural numbers, $\mathbb{R}$ for real numbers, $\mathbb{R}_{\geq 0}$ for non-negative reals, and $\mathbb{R}_{>0}$ for positive real numbers.

**Intervals.** An *interval* $I$ is of the form $(\!|t_1, t_2|\!)$ where $t_1, t_2 \in \mathbb{R} \cup \{\infty\}$ with $t_1 \leq t_2$ ($\leq$ is defined as expected for $\infty$), and $(\!| \in \{(, [\}$ and $|\!) \in \{), ]\}$. We will denote by $t + (\!|t_1, t_2|\!)$, the interval $(\!|t + t_1, t + t_2|\!)$. For $I = (\!|t_1, t_2|\!)$, $t_1$ and $t_2$ are the *left* (denoted $L(I)$) and *right* (denoted $R(I)$) *endpoints* of $I$, respectively. An interval is called *singular* if it is of the form $[t, t]$. Two intervals $I_1 = (\!|t_1, t_2|\!)$ and $I_2 = (\!|t_3, t_4|\!)$ are said to be *consecutive* if $t_2 = t_3$, $t_2$ is in exactly one of $I_1$ or $I_2$, and $I_1 \cap I_2 = \emptyset$. For example, $[1, 2)$ and $[2, 4)$ are consecutive intervals. An *interval sequence* is an finite sequence of intervals $I_1, I_2, I_3, \ldots, I_n$ that satisfies the following two conditions: [Initial] $L(I_1) = 0$ and $0 \in I_1$; and [Consecution] for each $i \geq 1$, $I_i$ and $I_{i+1}$ are consecutive intervals.

### A. Syntax

Formulas in HCMTL* are built using atomic propositions $P$, logic connectives, and modal operators. To allow one to reason about multiple executions, it has variables representing finite executions that can be quantified; $\mathcal{V}$ is the set of such *path variables*. The BNF grammar for formulas in HCMTL* is given below.

$$\varphi ::= p_\pi \mid \neg p_\pi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid F_I \varphi \mid G_I \varphi \mid \varphi U_I \varphi \mid \exists \pi \varphi \mid \forall \pi \varphi$$

In the grammar above, $p \in P$ is an atomic proposition, $\pi \in \mathcal{V}$ is a path variable, and $I$ is an interval. As in MTL, we allow singular intervals of the form $[t, t]$ to decorate modal operators, which distinguishes it from other logics that are based on MITL [5]. We use HyperMITL to denote the fragment of HCMTL* in which all quantifiers occur at the top-level of the formula and all intervals are non-singular.

Before defining the formal semantics for HCMTL*, we informally describe what formulas mean. The logic reasons about multiple finite executions of a real time system that are referred to by path variables. Atomic propositions capture abstract truths that may hold at different times during an execution. In order to distinguish between propositions in different executions, we annotated propositions with the path for which it is being asserted. Thus, $p_\pi$ asserts that proposition $p$ is true in execution $\pi$ currently. Similarly, $\neg p_\pi$ asserts that $p$ is false in $\pi$. It is important to recognize that here $\neg p_\pi$ asserts that $p$ is false, not that $p$ is not true. For example at a time $t$ that is after execution $\pi$ ends, neither $p_\pi$ nor $\neg p_\pi$ hold for $p$ is neither true nor false as $\pi$ has ended. Thus, the law of excluded middle does not hold in HCMTL*. As a consequence, negation needs to be handled carefully (see Remark 2) and consequently HCMTL* has more modal connectives than in typical presentations of a temporal logic.

Formulas can be combined using Boolean connectives conjunction ($\wedge$) and disjunction ($\vee$). The operators $F_I$, $G_I$, and $U_I$ express the modal and real-time aspects of the logic. These operators are real time extensions of the classical finally, globally, and until operators found in LTL [30] and CTL* [16], which constrain modal operators with time intervals requiring obligations to hold within those intervals. The formula $F_I \varphi$ (read 'finally $\varphi$') asserts that $\varphi$ holds at a time $t$ from the current time where $t$ is in the interval $I$. $G_I \varphi$ (read 'globally $\varphi$') asserts that $\varphi$ holds at all times $t$ from the current time when $t$ is in interval $I$. $\varphi_1 U_I \varphi_2$ (read '$\varphi_1$ until $\varphi_2$') holds if $\varphi_2$ holds at some time $t$ units from the current time where $t \in I$ and $\varphi_1$ holds continuously from the current time until $\varphi_2$ holds. For example, $\varphi_1 U_{[0,2]} \varphi_2$ says that eventually, within 2 units of time from now, $\varphi_2$ is true and $\varphi_1$ is true until then.

As mentioned before, HCMTL* expresses hyperproperties through path variables that stand for finite length executions, and quantifying over them. $\exists \pi \varphi$ asserts that there is an execution such that $\varphi$ holds, while $\forall \pi \varphi$ says that $\varphi$ holds no matter what execution is assigned to $\pi$. Like in other branching temporal logics for hyperproperties [12], when executions are quantified, the chosen execution for the variable is required to be an extension or branch of a 'current' execution; the 'current' execution depends on the context of the larger sentences in which a particular quantified sub-formula appears. This subtle aspect of quantification highlighted in Example 1. The variable $\pi$ is *bound* in the formulas $\exists \pi \varphi$ and $\forall \pi \varphi$. Any variable that does not appear within the scope of a quantifier is said to be *free*. Without loss of generality, we assume that a variable is bound at most once and does not appear both bound and free; these assumptions can be easily met by renaming bound variables. Finally, due to technical reasons that will become clear when we discuss the formal semantics, the modal operators $F_I$, $G_I$, and $U_I$ are required to always appear within the scope of quantifier.

*Example* 1. As in other branching temporal logics for hyper-

properties [12], quantified executions are required to be extensions or branches of certain other executions. We illustrate this through a couple of examples. The formula $\forall\pi_1 F_{[1,2]}\exists\pi_2\varphi$ says that for every execution $\pi_1$ of the system, within 1 to 2 units of time, there is an extension $\pi_2$ of $\pi_1$ that satisfies $\varphi$. In this case, the extension means that $\pi_1$ and $\pi_2$ must agree upto the point in time when the obligation for the operator $F$ is met. On the other hand, the formula $\forall\pi_1\exists\pi_2 F_{[1,2]}\varphi$ says that for every execution $\pi_1$ of the system, there exists an execution $\pi_2$ of the system such that within 1 to 2 units of time, $\pi_1$ and $\pi_2$ satisfy $\varphi$. Here, $\pi_1$ and $\pi_2$ are not required to agree (except at the very beginning) and can be thought to be completely independent executions.

*Remark* 2. It is typically convenient for logics to be closed under negation. Since negation in HCMTL* is restricted to only be applied to propositions, the logic has existential and universal quantifiers, and additional modal operators to ensure that negations can always be pushed inside. De Morgan's laws and the duality between exists and for all, allow one to handle the usual logical operators. For the modal connectives the following equivalences hold: $\neg F_I\varphi \equiv G_I\neg\varphi$, $\neg G_I\varphi \equiv F_I\neg\varphi$, and $\neg(\varphi_1 U_I \varphi_2) \equiv (G_I\neg\varphi_2) \vee (\neg\varphi_2 U_J \neg\varphi_1)$ where $J = [0, R(I))$.

### B. Timed Automaton

Timed automata [4] are a popular formal model to describe real time systems. They are an extension of finite automata that are equipped with clocks that can be individually reset and measure time since the last reset. Clocks can be used to enforce real time constraints during a system execution. All clocks in a timed automaton are *synchronous*, and progress in lockstep with an ambient global clock. We introduce this model in this section.

**Clock Constraints.** A clock constraint over a clock $x$ is formula given by the following grammar.

$$\psi \ ::= \ x \sim c \mid \psi \vee \psi \mid \psi \wedge \psi$$

where $c \in \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$. We will denote by $\Phi(x)$ the set of all clock constraints over $x$. For a set of clocks $X$, $\Phi(X)$ is the union of $\Phi(x)$ for all $x \in X$. A clock valuation is a map $\mu : X \to \mathbb{R}_{\geq 0}$. The satisfaction relation $\mu \models \psi$ is defined inductively as follows.

- $\mu \models x \sim c$ iff $\mu(x) \sim c$ is true.
- $\mu \models \psi_1 \vee \psi_2$ iff $\mu \models \psi_1$ or $\mu \models \psi_2$.
- $\mu \models \psi_1 \wedge \psi_2$ iff $\mu \models \psi_1$ and $\mu \models \psi_2$.

**Timed Automata.** A *timed automaton* over a set of atomic propositions $P$ is a tuple, $\mathcal{A} = (V, V_0, \alpha, X, \beta, E, V_F)$, where:

- $V$ is a finite set of states.
- $V_0 \subseteq V$ is a set of initial states.
- $\alpha : V \to 2^P$ is a state labeling function that labels each state with the set of propositions that are true at that state.
- $X$ is a finite set of clocks.
- $\beta : V \times X \to \Phi(X)$ is a function that labels each state, clock pair $(v, x)$ with a clock constraint over $x$.
- $E \subseteq V \times V \times 2^{\Phi(X)} \times 2^X$ is a set of transition edges of the automaton. An edge $(v_1, v_2, \Psi, \gamma)$ is a transition from state $v_1$ to $v_2$ that satisfies all the clock constraints in the guard $\Psi$ and resets the clocks in $\gamma \subseteq X$.
- $V_F \subseteq V$ is the set of final states.

**Runs/Executions.** An execution $\rho$ of $\mathcal{A}$ is a finite sequence

$$(v_1, I_1) \xrightarrow[\Psi_1]{\gamma_1} (v_2, I_2) \xrightarrow[\Psi_2]{\gamma_2} (v_3, I_3) \xrightarrow[\Psi_3]{\gamma_3} \dots \xrightarrow[\Psi_{n-1}]{\gamma_{n-1}} (v_n, I_n)$$

where $v_i \in V$ for all $i \in \{1, 2, \dots, n\}$ and $I_1, I_2, \dots, I_n$ is an interval sequence such that $v_1 \in V_0$, $(v_i, v_{i+1}, \Psi_i, \gamma_i) \in E$ for each $i$, and the real time constraints imposed in each state and transition are satisfied. To define when real time constraints are met, we need to define clock valuations at each time during the execution. We begin by first defining the clock valuations when first entering state $v_i$. Let the sequence of clock valuations $\mu_1, \mu_2, \dots \mu_n$ be inductively defined as follows: $\mu_1(x) = 0$ for all $x \in X$, and for all $i \geq 1$, $\mu_{i+1}(x) = \mu_i(x) + R(I_i) - L(I_i)$ if $x \notin \gamma_i$, and 0 otherwise. Next, for $t \in I_i$, the clock valuation at time $t$, $\mu_t$, is given as $\mu_t(x) = \mu_i(x) + t - L(I_i)$. Finally, for $\rho$ to be an execution the following two conditions must hold: [State Constraints] for every $i$, clock $x$ and $t \in I_i$, $\mu_t(x) \models \beta(v_i, x)$, and [Guard Constraints] for every $i$, clock $x$, and $\psi \in \Psi_i \cap \Phi(x)$, $\mu'(x) \models \psi$, where $\mu'(x) = \mu_i(x) + R(I_i) - L(I_i)$.

The execution $\rho$ is said to be *accepting* if $v_n \in V_F$. The collection of all accepting executions of $\mathcal{A}$ will be denoted by $\mathsf{exec}(\mathcal{A})$. We say $t \in |\rho|$ iff $t \in I_i$ for some $i$. An accepting execution $\rho$ is said to be *bounded by* $N \in \mathbb{R}$ if $t \notin |\rho|$ for all $t \geq N$; the set of all accepting executions bounded by $N$ of $\mathcal{A}$ will be denoted by $\mathsf{exec}_N(\mathcal{A})$. For $t \in |\rho|$, the state at time $t$, denoted $\rho(t)$, is $v_i$ if $t \in I_i$. The *prefix of $\rho$ up to time* $t \in |\rho|$, denoted $\rho|_t$, is the execution

$$(v_1, I_1) \xrightarrow[\Psi_1]{\gamma_1} (v_2, I_2) \xrightarrow[\Psi_2]{\gamma_2} (v_3, I_3) \xrightarrow[\Psi_3]{\gamma_3} \dots \xrightarrow[\Psi_{i-1}]{\gamma_{i-1}} (v_i, J_t)$$

where $t \in I_i$ and $J_t = I_i \cap [0, t]$. Two executions $\rho_1$ and $\rho_2$ are said to be *equal up to time* $t$ if $\rho_1|_t$ and $\rho_2|_t$ are identical.

### C. Semantics

We introduce the interval-based semantics here. Let us fix a timed automaton $\mathcal{A} = (V, V_0, \alpha, X, \beta, E, V_F)$. To define whether a HCMTL* formula $\varphi$ is true in $\mathcal{A}$ at a particular time $t$, we need to know what execution is assigned to each free path variable in $\varphi$. This is captured by a *path environment*. A path environment $\Pi : \mathcal{V} \to \mathsf{exec}(\mathcal{A})$ is a mapping that associates with each path variable in $\mathcal{V}$ an accepting execution of $\mathcal{A}$; we assume that the set of free variables of $\varphi$ is included in $\mathcal{V}$. When $\mathcal{V} = \emptyset$, the *empty path environment* is denoted by $\{\}$. For a path environment $\Pi$ over $\mathcal{V}$, a variable $\pi$, an execution $\rho \in \mathsf{exec}(\mathcal{A})$, $\Pi[\pi \mapsto \rho]$ denotes the path environment with domain $\mathcal{V} \cup \{\pi\}$ that is identical to $\Pi$, except that $\pi$ is now mapped to $\rho$. In branching hyperproperty logics like HyperCTL* [12], when a variable is quantified, it is expected to be assigned to an execution that is an extension/branch of an execution that is currently assigned to a variable (also see Example 1). Therefore, to define the semantics, we also need

to know which execution needs to be extended next at a future quantification. This is captured by † which takes values in $\mathcal{V} \cup \{\epsilon\}$; when $\dagger = \epsilon$, it indicates that the next quantified variable is completely independent. The satisfaction relation for HCMTL* captures when a formula $\varphi$ is true in a timed automaton $\mathcal{A}$ at time $t$ with respect to a path environment $\Pi$ and $\dagger \in \mathcal{V} \cup \{\epsilon\}$. It is denoted $\Pi, t, \dagger \models_{\mathcal{A}} \varphi$, and is defined inductively as follows.

- $\Pi, t, \dagger \models_{\mathcal{A}} p_\pi$ iff $t \in |\Pi(\pi)|$, and $p \in \alpha(\Pi(\pi)(t))$.
- $\Pi, t, \dagger \models_{\mathcal{A}} \neg p_\pi$ iff $t \in |\Pi(\pi)|$, and $p \notin \alpha(\Pi(\pi)(t))$.
- $\Pi, t, \dagger \models_{\mathcal{A}} \varphi_1 \vee \varphi_2$ iff $\Pi, t, \dagger \models_{\mathcal{A}} \varphi_1$ or $\Pi, t, \dagger \models_{\mathcal{A}} \varphi_2$.
- $\Pi, t, \dagger \models_{\mathcal{A}} \varphi_1 \wedge \varphi_2$ iff $\Pi, t, \dagger \models_{\mathcal{A}} \varphi_1$ and $\Pi, t, \dagger \models_{\mathcal{A}} \varphi_2$.
- $\Pi, t, \dagger \models_{\mathcal{A}} F_I \varphi$ iff there exists $t' > t$ such that $t' \in t + I$ and $\Pi, t', \dagger \models_{\mathcal{A}} \varphi$.
- $\Pi, t, \dagger \models_{\mathcal{A}} G_I \varphi$ iff for all $t' > t$, such that $t' \in t + I$, $\Pi, t', \dagger \models_{\mathcal{A}} \varphi$.
- $\Pi, t, \dagger \models_{\mathcal{A}} \varphi_1 U_I \varphi_2$ iff there exists $t' > t$ such that $t' \in t + I$ and $\Pi, t', \dagger \models_{\mathcal{A}} \varphi_2$, and for all $t < t'' < t'$, $\Pi, t'', \dagger \models_{\mathcal{A}} \varphi_1$.
- $\Pi, t, \dagger \models_{\mathcal{A}} \exists \pi \varphi$ iff there is an execution $\rho \in \mathsf{exec}(\mathcal{A})$ such that $\Pi[\pi \mapsto \rho], t, \pi \models_{\mathcal{A}} \varphi$ and either (a) $\dagger = \epsilon$ and $t = 0$, or (b) $t \in |\Pi(\dagger)|$ and $\rho|_t = \Pi(\dagger)|_t$.
- $\Pi, t, \dagger \models_{\mathcal{A}} \forall \pi \varphi$ iff for every execution $\rho \in \mathsf{exec}(\mathcal{A})$, if either (a) $\dagger = \epsilon$ and $t = 0$, or (b) $t \in |\Pi(\dagger)|$ and $\rho|_t = \Pi(\dagger)|_t$, then $\Pi[\pi \mapsto \rho], t, \pi \models_{\mathcal{A}} \varphi$.

**Bounded Time Semantics.** Bounded time semantics captures the notion that a timed automaton meets a specification $\varphi$ up to time $N \in \mathbb{R}_{\geq 0}$. This is captured by a satisfaction relation $\Pi, t, \dagger \models^N_{\mathcal{A}} \varphi$ which is defined in manner very similar to the definition above, except that $t$ is required to be $< N$, the path environment $\Pi : \mathcal{V} \to \mathsf{exec}_N(\mathcal{A})$ maps variables to accepting executions bounded by $N$ and whenever a new variable is quantified, it is assigned an execution in $\mathsf{exec}_N(\mathcal{A})$. The formal definition is skipped due to space constraints.

**Verification Problems.** This paper studies two decision problems associated with HCMTL*.

**[General]** Given a timed automaton $\mathcal{A}$ and an HCMTL* sentence $\varphi$, determine if $\{\}, 0, \epsilon \models_{\mathcal{A}} \varphi$.

**[Bounded Time]** Given a timed automaton $\mathcal{A}$, an HCMTL* sentence $\varphi$, and a time bound $N$, determine if $\{\}, 0, \epsilon \models^N_{\mathcal{A}} \varphi$.

## III. Examples

In this section, we highlight the expressive power of HCMTL* through examples. Security specifications in these examples demand reasoning about multiple executions, have real-time constraints, require analyzing the branching structure, and use different quantifiers when bounding variables. The security guarantees in the first three examples (timing attacks, secure multi-execution, opacity) are linear hyperproperties. The security guarantees in the last four examples (timed commitments, contract signing, unlinkability, and fair reward) are branching hyperproperties that cannot be expressed in real-time extensions of HyperLTL. Four examples involved quantifier alternation (opacity, contract signing, unlinkability, and fair reward). Two examples (timed commitment and contract signing) require non-trivial intervals to express the desired security guarantees. Finally, the unlinkability and fair reward examples are branching hyperproperties that relate executions along different branches and, thus, would not fall into branching time extensions of MTL. Thus the full power of HCMTL* is used to describe all the security requirements in these examples. Please note that in our examples below, we will use abbreviations like implication ( $\implies$ ) and equivalence ( $\iff$ ), which can be expressed in our logic by using the usual translation and pushing negations inside.

**Timing Attacks.** Programs computing over sensitive information should not be susceptible to leaking information through timing channels. To ensure that there are no such timing leaks, the program needs to guarantee that any two executions working on the same observable data (but possibly different private data) have the same timing behavior. Let $\mathcal{O}$ be set of observable inputs, and let the proposition $\mathsf{o}(a)$ for $a \in \mathcal{O}$ denote that the input $a$ is observed. Let run be the proposition to indicate that the program is running. Then using such propositions, the absence of timing leaks can be written as

$$\forall \pi_1 \forall \pi_2 . \Big( \bigwedge_{a \in \mathcal{O}} \mathsf{o}(a)_{\pi_1} \iff \mathsf{o}(a)_{\pi_2} \Big) \implies G_{[0,\infty)} (\mathsf{run}_{\pi_1} \iff \mathsf{run}_{\pi_2})$$

This formula says that for all paths $\pi_1$ and $\pi_2$, if they start with the same observable inputs, then globally they should run for the same time.

**Secure Multi-Execution (SME).** Non-interference requires that low-level (observable) outputs of two executions be the same if they are computing on the same low-level (observable) inputs. In other words, the difference in the high security inputs of two executions is not observable in the outputs. Secure multi-execution is an approach to ensure non-interference, where for any sequence of tasks, each task in the sequence is executed in two ways, one is a "low copy" and other is a "high copy". In the low copy, the high security inputs are set to some default values and the resulting outputs from this computation are observable. In the high copy, computation is carried with all the exact high security inputs, and the outputs from this computation are kept secure and non-observable. This ensures that any two executions operating on the same low-level inputs, have the same observations since only the outputs from the low copy are public which have default high security inputs. The low and high copies are interleaved for each task in the sequence. While SME ensures that the computation is non-interferent in the classical sense, it is open to timing attacks to an adversary observing the time duration between successive low copy computations if the high copy computations are of different length on inputs with the same low-level input. Such timing vulnerabilities can be described in HCMTL*. Let $\mathcal{O}$ to be a set of observable input values and $\mathsf{o}(a)$ for $a \in \mathcal{O}$ be the proposition that low-level input $a$ is observed. Let Hstart, Hrun, Hend be propositions denoting that high copy computation has started, is running, and has

ended, respectively. Timing vulnerability can be written as

$$\exists\pi_1\exists\pi_2.\ (\bigwedge_{a\in\mathcal{O}} \mathsf{o}(a)_{\pi_1} \iff \mathsf{o}(a)_{\pi_2})\ U_{[0,\infty)}\ \varphi$$

where

$$\varphi = \mathsf{Hstart}_{\pi_1} \wedge \mathsf{Hstart}_{\pi_2} \wedge ((\mathsf{Hrun}_{\pi_1} \wedge \mathsf{Hrun}_{\pi_2})\ U_{[0,\infty)}\ (\neg(\mathsf{Hend}_{\pi_1} \iff \mathsf{Hend}_{\pi_2})))$$

The formula says that there executions $\pi_1$ and $\pi_2$ that have the same low-level inputs until time $t$ when the formula $\varphi$ becomes true. $\varphi$ asserts that at time $t$ high copy computations start in $\pi_1$ and $\pi_2$ and the computation in either $\pi_1$ or $\pi_2$ ends before the other.

**Opacity.** Opacity of a property $\psi$ demands that the truth of $\psi$ be undeterminable to an adversary. This can be formalized by demanding that for every execution $\pi_1$ there is another execution $\pi_2$ that has the same observable behavior, but $\psi$ is true in exactly one of $\pi_1$ and $\pi_2$. To state this in HCMTL*, we use the following propositions: for a set of observations $\mathcal{O}$, the proposition $\mathsf{o}(a)$ asserts that $a \in \mathcal{O}$ is observed; end asserts that the computation has ended; and, $\psi$ is property of the last state expressed using a Boolean combination of some propositions. Then opacity of $\psi$ is

$$\forall\pi_1\exists\pi_2.\ G_{[0,\infty)}\ ((\textstyle\bigwedge_{a\in\mathcal{O}} \mathsf{o}(a)_{\pi_1} \iff \mathsf{o}(a)_{\pi_2}) \wedge ((\mathsf{end}_{\pi_1} \wedge \mathsf{end}_{\pi_2}) \implies \neg(\psi_{\pi_1} \iff \psi_{\pi_2})))$$

Notice that opacity requires alternation of path quantifiers in order to express it.

**Timed Commitment.** Consider the problem of tossing a coin when the *caller* (Alice) and the *tosser* (Bob) are in different locations but have a *reliable* communication channel to use. The setup is that Alice calls the toss, Bob tosses, and Alice wins if the result of the toss is what she predicts while Bob wins if it is not. A näive protocol might be that Alice sends her prediction to Bob and Bob then sends the result of the coin toss to Alice. At this point both parties know who the winner is. However, such a protocol is not fair to Alice as a dishonest Bob can always report a result of a coin toss that is the opposite of what Alice called. To circumvent this, Alice could *commit* her call, instead of sending it. Bob then tosses his coin and shares the result with Alice. At this point, Alice reveals her commitment and now both parties know the winner. However, such a protocol could be unfair to Bob, as a dishonest Alice may never reveal her commitment if she realizes that she lost. The solution that ensures fairness for both parties is to use a *timed commitment*, where the commitment is revealed to Bob after the elapse a fixed time $T$ even if Alice takes no steps towards revealing her commitment. The steps of such a protocol are as follows: Alice commits her call within time $t_c$, Bob has to share the result of the coin toss within time $T$ after Alice's commitment, and if he does, either Alice will reveal her call or Bob can compute what Alice committed to after $T$ units after Alice's commitment. If Bob does not toss a coin within time $T$, Alice is released from her commitment. Let us formally define the fairness guarantees for Alice and Bob, which are different since the protocol is asymmetric. Let the proposition $\mathsf{c}(b)$, for $b \in \{\mathsf{H}, \mathsf{T}\}$ denote that Alice has committed bit $b$, even though $b$ itself is not revealed to Bob. We assume that communication is reliable and so once Alice commits, Bob knows that she did. Proposition $\mathsf{t}(b)$ for $b \in \{\mathsf{H}, \mathsf{T}\}$ denotes a state when Bob has shared the result of a coin toss to be $b$. Finally $\mathsf{r}(b)$ asserts that Alice's commitment has been revealed to Bob and that call was $b$. Fairness for Bob now is

$$\forall\pi_1.\ G_{[0,t_c]} \bigwedge_{b\in\{\mathsf{H},\mathsf{T}\}} (\mathsf{c}(b)_{\pi_1} \implies (\forall\pi_2.\ F_{[T,\infty)}\mathsf{r}(b)_{\pi_2})).$$

It says for every execution $\pi_1$, if Alice commits within time $t_c$, then in every extension $\pi_2$ of $\pi_1$, Bob will be able to see Alice's commitment at some time that is $T$ units after Alice's commitment. In addition, Bob will see the same bit that Alice committed. Fairness for Alice can be written as

$$\forall\pi_1.\ G_{[0,t_c]}\ \ ((\mathsf{c}(\mathsf{H})_{\pi_1} \vee \mathsf{c}(\mathsf{T})_{\pi_1}) \implies \forall\pi_2.\ G_{[0,T]}((\neg\mathsf{t}(\mathsf{H})_{\pi_2} \wedge \neg\mathsf{t}(\mathsf{T})_{\pi_2}) \implies (\neg\mathsf{r}(\mathsf{H})_{\pi_2} \wedge \neg\mathsf{r}(\mathsf{T})_{\pi_2})))$$

which says that in every execution, if Alice commits within time $t_c$ then for the next $T$ units, if Bob has not yet shared the result of his coin toss, then Alice's commitment is not revealed. Notice that these properties are branching hyperproperties that cannot be expressed in linear hyperproperty logics even if they are real time.

**Contract Signing.** Consider a contract signing protocol mediated by a trusted third party (TTP) where two parties wish to sign a contract. Each signer must have the ability to move on in a timely manner if the other party does not complete the signing protocol. Each signer must be able to reach an abort state if they want or must receive a message from the TTP that the protocol is aborted in a timely manner. For $i \in \{1, 2\}$, let us consider the following propositions. $\mathsf{start}(i)$ means that party $i$ has started the protocol, $\mathsf{signed}(i)$ means that party $i$ has a signed contract, $\mathsf{abort}(i)$ means that party $i$ is in an abort state, and $\mathsf{token}(i)$ means that the TTP has provided an abort token to party $i$. We can now write the fairness for party $i$ as

$$\forall\pi_1.\ G_{[0,t]}\ (\mathsf{start}(i)_{\pi_1} \implies (\exists\pi_2.\ F_{[0,T]}(\mathsf{signed}(i)_{\pi_2} \vee \mathsf{abort}(i)_{\pi_2} \vee \mathsf{token}(i)_{\pi_2}))).$$

This is once again a branching hyperproperty and it also involves quantifier alternation.

**Unlinkability.** Radio Frequency Identification (RFID) is a technology used to identify and track physical objects using electromagnetic tags. A RFID system consists of tags attached to objects of interest and a reader that can communicate with the tags using electromagnetic waves to ascertain their identity and location. A security property that is required in such systems is unlinkability [11] i.e. when multiple rounds of communication happens between a tag and the reader, an adversary that is observing the communications should not be able to link the communications to the same tag. If this property is violated, an adversary can potentially track an object by linking all communications between that particular

tag and the receiver. Consider a simple RFID system with two tags. For $i \in \{1,2\}$, let $\mathsf{comm}(i)$ denote that tag $i$ communicated with the receiver. For a set $\mathcal{O}$ of observations, let the propositions $\mathsf{o}(a)$ denote that $a \in \mathcal{O}$ has been observed. We can state unlinkability of communications between tag 1 and the receiver in HCMTL* as

$$\forall\pi_1.(\mathsf{comm}(1)_{\pi_1} \implies G_{[0,\infty)}(\forall\pi_2\exists\pi_3.\, F_{(0,\epsilon)}(\mathsf{comm}(1)_{\pi_2}) \implies F_{(0,\epsilon)}(\mathsf{comm}(2)_{\pi_3}) \wedge \varphi)).$$

where $\epsilon > 0$ and

$$\varphi = G_{[0,\infty)}(\bigwedge_{a\in\mathcal{O}} \mathsf{o}(a)_{\pi_2} \iff \mathsf{o}(a)_{\pi_3}).$$

It says that for any execution $\pi_1$ where tag 1 communicates with the receiver initially, globally, for all branches $\pi_2$ where tag 1 communicates with the receiver, there is another branch $\pi_3$ where tag 2 communicates with the receiver, such that $\pi_2$ and $\pi_3$ have the same observable behaviour to an adversary. This ensures that the adversary cannot link the two communications between tag 1 and the receiver. Observe that this is a branching *hyperproperty* that relates two traces, and hence would not be captured by a branching time extension of MTL.

**Fair Reward.** Consider a distributed program running on top of a blockchain based cryptocurrency as described in [31]. The system consists of users that submit transactions in the form of contracts to the program, which are then executed and published by the program to the blockchain. In between a transactions submission and publication, it becomes public to other users, who might then choose to submit new transactions based on this. Note that we are considering a distributed program running on top of the blockchain and hence the submissions are not necessarily published in order of submission. The only requirement of the program is that submissions are eventually published or returned as invalid contracts. Consider a simple such model where a user if rewarded (monetarily) for correctly computing the pre-image of some hashed value. In such a system, after an honest user submits a correct answer (in the form of a transaction), an adversary might use the information from the transaction to submit their own answer and if it gets published first, the adversary can steal a promised reward from the honest user. The fair reward property requires that the program should not be vulnerable to such attacks. Formally, for any execution of the system, there is another execution (called ideal execution) where submissions are published in order and the final balances of all users in both executions are equal. The protocol allows a setup phase for the adversary where the adversary chooses the attack parameters. The proposition $\mathsf{setup}$ indicates the end of the setup phase. The fair reward property now requires that in any execution of the system, once the adversary is finished setting up, there is an ideal branch of the execution such that both executions have the same balance for all users eventually. For a set $\mathcal{T}$ of transactions (each transaction can be thought of as a bit string) and a transaction $x \in \mathcal{T}$, let $\mathsf{submit}(x)$ and $\mathsf{publish}(x)$ denote that $x$ has been submitted and published respectively. Let $\mathsf{silent}$ denote that no transaction has been submitted or published. In the remainder of this example, to avoid clutter, we will use the symbols $F, G, U$ without any interval annotation to indicate that the interval is $[0,\infty)$. An ideal execution can be modeled in HCMTL* as

$$\mathsf{ideal}(\pi) = G\,(\bigwedge_{x\in\mathcal{T}} \mathsf{submit}(x)_\pi \wedge F\,\mathsf{publish}(x)_\pi \implies \mathsf{silent}_\pi U \mathsf{publish}(x)_\pi).$$

This says that globally, if a transaction is submitted and eventually published, then the execution is silent until the publication. Let $\mathcal{B}$ be the set of all possible balance profiles (these can be thought of as bit strings indicating the balance of each user), and for $b \in \mathcal{B}$, let $\mathsf{balance}(b)$ be the proposition indicating that the balance profile is described by $b$. Fair reward can now be modeled in HCMTL* as

$$\begin{aligned}\forall\pi_1.\, G(&\mathsf{setup}_{\pi_1} \implies \\ &\exists\pi_2.\,\mathsf{ideal}(\pi_2) \wedge \\ &(\bigwedge_{x\in\mathcal{T}} F\,\mathsf{submit}(x)_{\pi_1} \iff F\,\mathsf{submit}(x)_{\pi_2}) \wedge \\ &F\,(G\,(\mathsf{silent}_{\pi_1} \wedge \mathsf{silent}_{\pi_2}) \wedge \\ &\quad \bigwedge_{b\in\mathcal{B}} \mathsf{balance}(b)_{\pi_1} \iff \mathsf{balance}(b)_{\pi_2}))\end{aligned}$$

This says that for every execution $\pi_1$, globally, once the adversary is done setting up, there is a branch $\pi_2$ which is an ideal execution, and the submissions in $\pi_1$ and $\pi_2$ are identical, and eventually, when both executions become silent, they have the same balance profiles.

## IV. Verifying HCMTL*

In this section we present the main results related to the verification problems for HCMTL* introduced in Section II-C. We show that the general verification is undecidable, but the bounded time verification problem is decidable. All the results we present in this section are for interval-based semantics.

**Theorem 3.** *The general verification problem for HCMTL* is undecidable in the interval-based semantics. In fact the verification problem is undecidable even for the fragment* HyperMITL.

The proof of Theorem 3 is deferred to Section V.

**Theorem 4.** *The bounded time verification problem for HCMTL* is decidable in the interval-based semantics.*

Our decidabiliy result is established by reducing the bounded time verification problem for HCMTL* to the satisfiability problem for Monadic Second Order logic with $<$ and $+1$ relations, denoted MSO$(<,+1)$, over a bounded time domain. The satisfiability problem for MSO$(<,+1)$ is decidable over bounded time domains [28], and our result thus follows. The following subsections outline the decidability proof and it is organized as follows.

1) First we introduce MSO$(<,+1)$ and state the relevant decidability results.
2) The first technical result in our proof shows that for any timed automaton $\mathcal{A}$, there is an MSO$(<,+1)$ formula $\varphi_{\mathcal{A}}$ whose models are exactly the accepting executions of $\mathcal{A}$.

3) Finally, using this translation of a timed automaton to an MSO$(<,+1)$ formula, we reduce the bounded time verification problem to the satisfiability problem of MSO$(<,+1)$.

### A. Monadic Second Order Logic

Monadic Second Order logic with $<$ and $+1$, denoted MSO$(<,+1)$, is built over a set of monadic predicates $\mathsf{MP}$ and set of first order variables $\mathsf{Vars}$. The BNF grammar is as follows.

$$\varphi \ ::= \ x < y \mid \ +1(x,y) \mid P(x) \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists x\varphi \mid \exists P\varphi$$

In the grammar above, $P \in \mathsf{MP}$ is a monadic predicate, and $x \in \mathsf{Vars}$ is a first order variable.

The semantics of MSO$(<,+1)$ is defined over a timed domain $\mathbb{T}$. We will define it over two time domains $\mathbb{T} = \mathbb{R}_{\geq 0}$ and $\mathbb{T} = [0, N)$ for some fixed $N \in \mathbb{N}$. Let $\mathsf{P} \subseteq \mathsf{MP}$ be the set of free monadic predicates in an MSO$(<,+1)$ formula $\varphi$. A *flow* is a map, $f : \mathbb{T} \to 2^{\mathsf{P}}$ that is *finitely variable* (explained next). For any interval $I \subseteq \mathbb{T}$, define the map $f|_I : I \to 2^{\mathsf{P}}$ as $f|_I(t) = f(t)$ i.e. it is the restriction of $f$ to the interval $I$. Now, we say a map $f : \mathbb{T} \to 2^{\mathsf{P}}$ is *finitely variable* if for any bounded interval $I \subseteq \mathbb{T}$ with finite endpoints, $f|_I$ has finitely many discontinuities i.e. the values of the monadic predicates change finitely many times in $I$. For $\mathsf{Q} \subseteq \mathsf{P} \subseteq \mathsf{MP}$, and a flow $g : \mathbb{T} \to 2^P$, we will denote by $g|_{\mathsf{Q}} : \mathbb{T} \to 2^Q$ the flow defined as $g|_{\mathsf{Q}}(t) = g(t) \cap \mathsf{Q}$. An interpretation $I$ is a map $I : \mathsf{Vars} \to \mathbb{T}$. We will denote by $I[x \mapsto a]$ the interpretation that maps $x$ to $a \in \mathbb{T}$, and is same as $I$ for all $y \neq x$. The semantics, denoted by $f, I \models \varphi$, is defined as follows.

1) $f, I \models x < y$ iff $I(x) < I(y)$.
2) $f, I \models +1(x,y)$ iff $I(y) = I(x) + 1$.
3) $f, I \models P(x)$ iff $P \in f(I(x))$.
4) $f, I \models \varphi_1 \vee \varphi_2$ iff $f, I \models \varphi_1$ or $f, I \models \varphi_2$.
5) $f, I \models \neg\varphi$ iff $f, I \not\models \varphi$.
6) $f, I \models \exists x\varphi$ iff there exists $a \in \mathbb{T}$ such that $f, I[x \to a] \models \varphi$.
7) $f, I \models \exists Q\varphi$ iff there is some finitely variable flow $g : \mathbb{T} \to 2^{\mathsf{P} \cup \{Q\}}$ such that $g|_{\mathsf{P}} = f$ and $g, I \models \varphi$.

**Satisfiability Problem.** Given an MSO$(<,+1)$ formula $\varphi$ over a set $\mathsf{MP}$ of free monadic predicates and free first order variables $\mathsf{Vars}$, determine if there is a flow $f : \mathbb{T} \to 2^{\mathsf{MP}}$ and $I : \mathsf{Vars} \to \mathbb{T}$ such that $f, I \models \varphi$.

When the time domain is bounded, i.e., $\mathbb{T} = [0, N)$ for some $N \in \mathbb{R}_{>0}$, the satisfiability problem is decidable [28].

**Theorem 5** ( [28]). *For $\mathbb{T} = [0, N)$ ($N \in \mathbb{R}_{>0}$), the satisfiability problem for MSO$(<,+1)$ is decidable.*

### B. Translating Timed Automata to MSO$(<,+1)$

We will now show that for any timed automaton $\mathcal{A}$, there is an MSO$(<,+1)$ formula $\varphi_{\mathcal{A}}$ that is satisfied exactly by the accepting executions of $\mathcal{A}$. The first challenge in this translation is that MSO$(<,+1)$ models are flows, which are functions from $\mathbb{T}$ to a set of monadic predicates, which are different from executions of a timed automaton. Hence, to formally state our result, we need to construct a one-to-one correspondence between executions of a timed automaton and flows.

Recall that an execution $\rho$ of a timed automaton $\mathcal{A} = (V, V_0, \alpha, X, \beta, E, V_F)$ is of the form

$$(v_1, I_1) \xrightarrow[\Psi_1]{\gamma_1} (v_2, I_2) \xrightarrow[\Psi_2]{\gamma_2} (v_3, I_3) \xrightarrow[\Psi_3]{\gamma_3} \dots \xrightarrow[\Psi_{n-1}]{\gamma_{n-1}} (v_n, I_n)$$

We will encode this as a flow $f : \mathbb{R}_{\geq 0} \to 2^V$ over the set of monadic predicates $\mathsf{MP} = V$ as follows:

- If $t \in I_i$, $f(t) = \{v_i\}$
- If $t \notin I_i$ for any $i$, $f(t) = \emptyset$ which indicates that the run has terminated and hence no state is present in the flow.

This is not sufficient because this does not give a one-to-one mapping from executions to flows. As an example, consider the following two simple executions:

$$\rho_1 = (v, [0, 10])$$
$$\rho_2 = (v, [0, 5]) \xrightarrow[\Psi]{\gamma} (v, (5, 10]).$$

These executions are different because one has a transition at time 5 while the other does not. However, if we think of them as a function $f : [0, 10] \to V$, they are identical. This indicates that in the function $f$, we also need to carry information about the transitions and the clock resets that occur during an execution. This involves some challenging subtleties that we illustrate with an example. Consider the following execution with two clocks $x_1$ and $x_2$,

$$\rho = (v_1, [0, 5)) \xrightarrow[\Psi_1]{\{x_1\}} (v_2, [5, 10]) \xrightarrow[\Psi_2]{\{x_2\}} (v_3, (10, 12))$$
$$\xrightarrow[\Psi_3]{\{x_2\}} (v_1, [12, 12]) \xrightarrow[\Psi_4]{\{x_1\}} (v_4, (12, 15))$$

Observe that at time 5, a transition $e_1 = (v_1, v_2, \Psi_1, \{x_1\})$ occurs and the automaton is in the target state of the transition i.e. state $v_2$. Let us call this type of transition a $T^-$ transition. At time 10 on the other hand, a transition $e_2 = (v_2, v_3, \Psi_2, \{x_2\})$ occurs and the automaton is in the source state of the transition i.e. state $v_2$, which we will call a $T^+$ transition. Finally, at time 12, both kinds of transitions occur. Thus, in the function $f$, we will define $f(5) = \{v_2, T^-_{e_1}\}$ to say that the execution is at state $v_2$ and a transition $e_1$ of type $T^-$ occurred. Similarly we can define $f(10) = \{v_2, T^+_{e_2}\}$ and at time 12, and $f(12)$ will contain the location $v_1$ and one $T^-$ and $T^+$ transitions. Corresponding to each transition, some clocks are reset. Now in a singular interval, for example time 12 in the execution above, $x_2$ is reset at the transition $(v_3, v_1, \Psi_3, \{x_2\})$ and then $x_1$ is reset in the transition $(v_1, v_4, \Psi_4, \{x_1\})$. Hence, we also need to differentiate clock resets into two types, resets associated to $T^-$ transitions and those associated to $T^+$ transitions. So for each clock $x \in X$, we will have two monadic predicates $x^-$ and $x^+$. When a $T^+$ transition corresponding to edge $e = (v_1, v_2, \Psi, \gamma)$ occurs at time $t$, we will add the resets $x^-$ for $x \in \gamma$ to $f(t)$, and similarly for $T^+$ transitions. The corresponding flow for the execution above will be:

- $f(t) = \{v_1\}$ for $t \in [0, 5)$
- $f(5) = \{v_2, T^-_{e_1}, x^-_1\}$
- $f(t) = \{v_2\}$ for $t \in (5, 10)$
- $f(10) = \{v_2, T^+_{e_2}, x^+_2\}$
- $f(t) = \{v_3\}$ for $t \in (10, 12)$
- $f(12) = \{v_1, T^-_{e_3}, x^-_2, T^+_{e_4}, x^+_1\}$
- $f(t) = \{v_4\}$ for $t \in (12, 15)$

This information is sufficient to ensure a one-to-one correspondence between executions and their corresponding flows. Using this idea, formally, we have the lemma

**Lemma 6.** *For a timed automaton $\mathcal{A} = (V, V_0, \alpha, X, \beta, E, V_F)$, let $T = \{T^-_e, T^+_e \mid e \in E\}$ and $R = \{x^-, x^+ \mid x \in X\}$. Define a set of monadic predicates $\mathsf{MP} = V \cup T \cup R$. Define $\mathcal{F}$ to be the set of all flows $f : \mathbb{R}_{\geq 0} \to 2^{\mathsf{MP}}$. There is a one-to-one encoding $F_{\mathcal{A}} : \mathsf{exec}(\mathcal{A}) \to \mathcal{F}$ of executions $\rho$ of $\mathcal{A}$ as flows.*

If we consider bounded executions of $\mathcal{A}$ for some time bound $N \in \mathbb{R}_{>0}$, the same encoding gives an encoding of bounded executions as flows $f : [0, N) \to 2^{\mathsf{MP}}$.

Now, given a timed automaton $\mathcal{A}$, we want to construct an MSO$(<, +1)$ formula $\varphi_{\mathcal{A}}$ over the monadic predicates $\mathsf{MP} = V \cup T \cup R$ such that $\varphi_{\mathcal{A}}$ is satisfied exactly by the set $F_{\mathcal{A}}(\mathsf{exec}(\mathcal{A}))$. The key idea behind this is that all the properties of an execution can be expressed in MSO$(<, +1)$. We illustrate how certain important properties of an execution can be expressed in MSO$(<, +1)$.

The most basic property of an execution is that it terminates at some time $l$ and at any time $t$ up to time $l$, the execution is exactly in one state of the automaton. First we have a formula $US(t)$ ($US$ stands for Unique state) which states that there is exactly one state at time $t$.

$$US(t) = \bigvee_{v \in V} v(t) \wedge \bigwedge_{v_1, v_2 \in V : v_1 \neq v_2} \neg(v_1(t) \wedge v_2(t))$$

Now, using this, we can express the above property as a disjunction of the following two formulas

$$\exists l \forall t((t \leq l \implies US(t)) \wedge (t > l \implies \wedge_{v \in V} \neg v(t)))$$
$$\exists l \forall t((t < l \implies US(t)) \wedge (t \geq l \implies \wedge_{v \in V} \neg v(t)))$$

The first one says that for time $t$ up to and including time $l$, there is exactly one state and after $l$ there is no state. The second one says that for time $t$ up to but not including $l$, there is exactly one state and for time $t \geq l$, there is no state. In a similar manner we also say that at all times where the execution is in some state, at most one $T^-_e$ and one $T^+_e$ predicate is true.

Another important property of an execution is that if a transition $e \in E$ of type $T^-$ occurs at time $t$, then the predicate $T^-_e$ should be true at time $t$ and vice versa. For each $e = (v_1, v_2, \Psi, \gamma)$ we have the formula

$$\forall t(T^-_e(t) \implies v_2(t) \wedge \exists y(y < t \wedge \forall z(y < z < t \implies v_1(t)))).$$

This ensures that if $T^-_e$ is true at time $t$, then there is transition of the form

$$(v_1, (\!|t_1, t)\!) \xrightarrow[\Psi]{\gamma} (v_2, [t, t_2|\!])$$

at $t$. For the opposite direction, i.e. to ensure that $T^-_e$ is true only at times $t$ where a corresponding transition $e$ occurs, for each pair of states $v_1 \neq v_2$, let $C$ be the set of all edges of the form $e = (v_1, v_2, \Psi, \gamma)$. We have the formula

$$\forall t(v_2(t) \wedge \exists y(y < t \wedge \forall z(y < z < t \implies v_1(t))) \implies \bigvee_{e \in C} T^-_e(t) \wedge \bigwedge_{e_1, e_2 \in C : e_1 \neq e_2} \neg(T^-_{e_1} \wedge T^-_{e_2}))$$

This formula says that if for a small open interval $(y, t)$ the automaton is in state $v_1$ and at time $t$ it enters state $v_2$, then exactly one transition of the form $(v_1, v_2, \Psi, \gamma)$ must have occurred at time $t$. An important point to note here is that the second formula is only for edges where $v_1 \neq v_2$. This is because if $v_1 = v_2$, i.e. the edge is a self loop, then it is not necessary for a transition to occur. The automaton may simply remain in state $v_1$ without making a transition at time $t$. We can write similar formulas for transition of type $T^+$.

We also need to ensure that clock resets happen only when transitions occur. For this, first we ensure that if no transition occurs at time $t$, then no clocks are reset at that time.

$$\forall t(\neg(\vee_{e \in E} T^-_e(t)) \implies \neg(\vee_{x \in X} x^-(t)))$$

For the other direction, we say that if a transition $e = (v_1, v_2, \Psi, \gamma)$ of type $T^-$ occurs, then the corresponding clock resets of type $x^-$ occur.

$$\forall t(T^-_e(t) \implies \wedge_{x \in \gamma} x^-(t) \bigwedge \wedge_{x \notin \gamma} \neg x^-(t))$$

Another property of an execution is that any time the clock constraints of the state is satisfied. To do this, we need to compute the value of a clock at any time $t$ of the execution. We cleverly use the clock reset predicates $R$ to find this value, and then compare it with the clock constraints of the current state. For example, if at time $t$ no transition occurs, then the value of a clock is the time elapsed since the last time it was reset. We can encode this as follows for each state $v \in V$ and clock $x \in X$:

$$\forall t(\neg(\bigvee_{e \in E} T^-_e(t) \vee T^+_e(t)) \wedge v(t) \implies \exists r(r < t \wedge (x^-(r) \vee x^+(r)) \wedge \forall z(r < z < t \implies \neg(x^-(z) \vee x^+(z))) \wedge \beta(v, x)[t - r])$$

Here, $\beta(v, x)[t - r]$ is the clock constraint $\beta(v, x)$ with $x$ replaced by $t - r$. This formula says that for all time $t$, if no transition occurs at $t$ and the execution is at location $v$, then there must exist a time $r$ before $t$, such that clock $x$ was reset at $r$ and for all time $z$ between $r$ and $t$, $x$ was not reset. Thus, $r$ is the last time clock $x$ was reset. Finally, we require that $t - r$ (i.e., the current value of clock $x$) satisfies the clock constraint $\beta(v, x)$. The satisfaction of guards for transitions can also be ensured using the same idea.

A point to note here is that $\beta(v,x)[t-r]$ has inequalities of the form $t - r \sim c$ where $\sim \in \{<, \leq, =, \geq, >\}$. The logic MSO$(<,+1)$ does not have constants, so we cannot directly write formulas of the form $t - r \sim c$. However, we can still express this in the following manner:

- For any constant $c$, we will write a formula $+c(r,x)$ such that $f, I \models +c(r,x)$ iff $I(x) = I(r) + c$. For $c = 1$, we can write this directly in MSO$(<,+1)$ as $+1(r,x)$
- Next, using the $+1$ relation, we can write $+2$ as follows:

$$+2(r,x) = \exists y(+1(r,y) \wedge +1(y,x)).$$

  Similarly, we can write $+3$ as:

$$+3(r,x) = \exists y(+2(r,y) \wedge +1(y,x)).$$

  In this manner, we can express every $+c(r,x)$ for any constant $c$ using finite sized formulas in MSO$(<,+1)$.
- Now we can rewrite $t - r \sim c$ as $t \sim r + c$. This we can write in MSO$(<,+1)$ as

$$t \sim r + c \equiv \exists x(+c(r,x) \wedge t < x)$$

In this manner, we can express all the properties of an execution in MSO$(<,+1)$. Formally, we have the following lemma:

**Lemma 7.** *Given a timed automtaton $\mathcal{A} = (V, V_0, \alpha, X, \beta, E, V_F)$, there is a MSO$(<,+1)$ formula $\varphi_{\mathcal{A}}$ over free monadic predicates $\mathsf{MP} = V \cup T \cup R$, such that*

1) *For any execution $\rho \in \mathsf{exec}(\mathcal{A})$, $F_{\mathcal{A}}(\rho) \models \varphi_{\mathcal{A}}$.*
2) *For any flow $f \models \varphi_{\mathcal{A}}$, there exists an execution $\rho \in \mathsf{exec}(\mathcal{A})$ such that $f = F_{\mathcal{A}}(\rho)$.*

The lemma also holds if we consider bounded time semantics with a time bound $N$.

### C. Decidability

We now give a reduction from the bounded time verification problem for HCMTL* to the satisfiability problem for MSO$(<, +1)$ over bounded time domains.

Fix $N \in \mathbb{R}_{>0}$ and a timed automaton $\mathcal{A} = (V, V_0, \alpha, X, \beta, E, V_F)$. Let $\varphi$ be an HCMTL* formula with free path variables $\mathcal{V}$. Let the variables in $\mathcal{V}$ be ordered as $\{\pi_1, \pi_2, \ldots, \pi_m\}$. Observe that the models to an HCMTL* formula are path environments while the models to an MSO$(<, +1)$ formula are flows. To reconcile this disparity, we use the encoding of executions as flows from Lemma 6. For a path environment $\Pi : \mathcal{V} \to \mathsf{exec}_N(\mathcal{A})$, we want to encode $\Pi$ as a flow. We do this by combining the flows corresponding to each execution $\Pi(\pi_i)$ into one single flow and distinguishing the predicates for each execution by indexing. For each path $\pi_i \in \mathcal{V}$, we define copies of the sets $V, T$ and $R$ as $V_i = \{v_i \mid v \in V\}$, and analogously for $T_i$ and $R_i$ . Define $\mathsf{MP}_i = \{V_i \cup T_i \cup R_i\}$ for each $\pi_i \in \mathcal{V}$. For each path variable $\pi_i$, $\Pi(\pi_i)$ is an execution in $\mathsf{exec}_N(\mathcal{A})$ and $F_{\mathcal{A}}(\Pi(\pi_i))$ is a flow over the monadic predicates $\mathsf{MP}_i$. Define $\mathsf{MP} = \cup_{\pi_i \in \mathcal{V}} \mathsf{MP}_i$. We can lift the encoding $F_{\mathcal{A}}$ to $\Pi$ by defining a flow $f^{\Pi} : [0, N) \to 2^{\mathsf{MP}}$ as:

$$f^{\Pi}(t) = \cup_{\pi_i \in \mathcal{V}} F_{\mathcal{A}}(\Pi(\pi_i))(t)$$

Our goal is to construct an MSO$(<,+1)$ formula ${}^{\mathcal{A}}T^{\varphi}$ that is satisfied exactly by the flows that encode path environments that satisfy $\varphi$. One of the challenges in constructing ${}^{\mathcal{A}}T^{\varphi}$ is handling the last quantified path †. Since HCMTL* is a logic for branching hyperproperties, the semantics of HCMTL* involves the last quantified path, that we represent using the variable †. To construct ${}^{\mathcal{A}}T^{\varphi}$, we need to encode the last quantified path into ${}^{\mathcal{A}}T^{\varphi}$. One option is to encode † as a first order variable in ${}^{\mathcal{A}}T^{\varphi}$ that takes values in $\{1, 2, \ldots, m\}$. However, this does not work because by the semantics of MSO$(<,+1)$, † can only take values in $[0, N)$. Hence, if the number of free path variables, $m$ is larger than $N$, it will not be possible to cover all values of †.

We overcome this by instead having $m+1$ formulas indexed ${}^{\mathcal{A}}T^{\varphi}_i$ for $i \in \{0, 1, 2, \ldots, m\}$. And the property that we preserve is the following: for any path environment $\Pi$ and $t \in [0, N)$,

$$\Pi, t, \pi_i \models^N_{\mathcal{A}} \varphi \text{ iff } f^{\Pi} \models {}^{\mathcal{A}}T^{\varphi}_i(t).$$

If $\dagger = \epsilon$, then the property we preserve is

$$\Pi, 0, \epsilon \models^N_{\mathcal{A}} \varphi \text{ iff } f^{\Pi} \models {}^{\mathcal{A}}T^{\varphi}_0(0).$$

We do this by constructing the formulas ${}^{\mathcal{A}}T^{\varphi}_i$ inductively. Translating quantifier free formulas to MSO$(<,+1)$ is straightforward and is identical for all $i \in \{0, 1, 2, \ldots, m\}$. For example, if $\varphi = p_{\pi_j}$, this means that at time $t$, $\pi_j$ is in a state where the proposition $p$ is true. The corresponding MSO$(<,+1)$ formula is:

$${}^{\mathcal{A}}T^{\varphi}_i(x) = \bigvee_{v \in V : p \in \alpha(v)} v_j(x)$$

Another example is $\varphi = \varphi_1 U_I \varphi_2$. This formula says that at some future time $y > t$, such that $y \in t + I$, the formula $\varphi_2$ must hold, and for all time $t < z < y$, the formula $\varphi_1$ holds. This naturally translates to MSO$(<,+1)$ as

$$\begin{aligned} {}^{\mathcal{A}}T^{\varphi}_i(x) = \exists y(x < y \wedge {}^{\mathcal{A}}T^{\varphi_2}_i(y) \wedge \\ \forall z(x < z < y \implies {}^{\mathcal{A}}T^{\varphi_1}_i(z)) \wedge y - x \in I) \end{aligned}$$

The construction for the $\exists$ and $\forall$ quantifiers involves the translation of timed automata to MSO$(<,+1)$ that we described in lemma 7. Suppose $\varphi = \exists \pi_{m+1} \varphi_1$. The corresponding MSO$(<,+1)$ formula ${}^{\mathcal{A}}T^{\varphi}_i$ should read 'there exists an execution that is identical to $\pi_i$ up to the current time, and the path environment $\Pi$ augmented with this new execution satisfies ${}^{\mathcal{A}}T^{\varphi_1}_{m+1}$.' Since we are quantifying over executions of the automaton, in ${}^{\mathcal{A}}T^{\varphi}$, we quantify over flows that satisfy the formula $\varphi_{\mathcal{A}}$ from Lemma 7, and this can only be done in a second order logic. The formula ${}^{\mathcal{A}}T^{\varphi}_i$ is constructed as:

$$\begin{aligned} {}^{\mathcal{A}}T^{\varphi}_i(x) = \exists V_{m+1}, T_{m+1}, R_{m+1} \\ (\varphi_{\mathcal{A}}(V_{m+1}, T_{m+1}, R_{m+1}) \\ \wedge \forall y(0 \leq y \leq x \implies (\mathsf{MP}_i(y) \iff \mathsf{MP}_{m+1}(y))) \\ \wedge {}^{\mathcal{A}}T^{\varphi_1}_{m+1}(x)) \end{aligned}$$

Here $\mathsf{MP}_i(y) \iff \mathsf{MP}_{m+1}(y)$ is an abbreviation for

$$\wedge_{p \in \mathsf{MP}}(p_i(y) \iff p_{m+1}(y))$$

i.e. the $i$th copy and $m+1$st copy of the predicates have the same truth values at $y$.

Putting these ideas together, we have the following lemma.

**Lemma 8.** *Fix $N \in \mathbb{R}_{>0}$, an HCMTL* formula $\varphi$ and a timed automaton $\mathcal{A} = (V, V_0, \alpha, X, \beta, E, V_F)$. There exist $m+1$ MSO$(<,+1)$ formulas ${}^{\mathcal{A}}T_0^{\varphi}(x), {}^{\mathcal{A}}T_1^{\varphi}(x), \ldots, {}^{\mathcal{A}}T_m^{\varphi}(x)$, each over MP with one free first order variable $x$ such that, for $t \in [0, N)$ and a path environment $\Pi$,*

1) *For $i \in \{1, 2, \ldots, m\}$, $\Pi, t, \pi_i \models^N_{\mathcal{A}} \varphi$ iff $f^{\Pi} \models {}^{\mathcal{A}}T_i^{\varphi}(t)$*
2) *$\Pi, 0, \epsilon \models^N_{\mathcal{A}} \varphi$ iff $f^{\Pi} \models {}^{\mathcal{A}}T_0^{\varphi}(0)$.*

We thus have this corollary

**Corollary 9.** *For an HCMTL* sentence $\varphi$, $\{\}, 0, \epsilon \models^N_{\mathcal{A}} \varphi$ iff ${}^{\mathcal{A}}T_0^{\varphi}(0)$ is a valid MSO$(<,+1)$ sentence over $\mathbb{T} = [0, N)$.*

Thus, by the corollary, time bounded verification of an HCMTL* sentence $\varphi$ in the interval-based semantics reduces to checking satisfiability of the MSO$(<,+1)$ formula ${}^{\mathcal{A}}T_0^{\varphi}(0)$ over the bounded time domain $[0, N)$, which is known to be decidable [28]. Thus, we get our main result

**Theorem 10.** *Bounded Time verification problem of HCMTL* is decidable in the interval-based semantics.*

*Remark* 11. In our presentation, we have restricted clock constraints in timed automata to allow only comparisons with natural numbers, and the intervals $I$ in temporal operators such as $U_I$ to have natural number end points. This is done for the sake of simplicity and all the results presented here carry over if we allow non-negative rational numbers instead of natural numbers. In case of rational bounds, the model checking problem can be reduced to a model checking problem with only natural number bounds by appropriately scaling all constants appearing the the timed automata and the HCMTL* formula. The appropriate scaling factor will be the least common multiple of all denominators occurring in all the rational constants.

## V. Point-based Semantics

We define a point-based semantics for our logic HCMTL*. We first give a point-based semantics for timed automata, then move on to HCMTL*, and finally present our results for the point based semantics.

### A. Timed Automata

In the interval-based semantics, the system is under observation at all times. On the other hand, in the point-based semantics, the system is observed at discrete time points when events (marked by propositions that are true at the event) occur. A timed automata in the point based semantics over a set of propositions $P$ is a tuple $\mathcal{B} = (2^P, S, s_0, X, \Delta, F)$ where

- $S$ is a finite set of states.
- $s_0 \in S$ is the start state.
- $X$ is a finite set of clocks.
- $\Delta \subseteq S \times 2^P \times 2^{\Phi(X)} \times 2^X \times S$ is the transition relation. A transition $e = (s_1, \sigma, \Psi, \gamma, s_2)$ is a transition from state $s_1$ to $s_2$ on event $\sigma$ that satisfies the guard $\Psi$ and resets the clocks in $\gamma$.
- $F \subseteq S$ is the set of final states.

For a transition $e = (s_1, \sigma, \Psi, \gamma, s_2)$, we will call $\sigma$ the event labelling $e$. An execution of $\mathcal{B}$ is a finite sequence

$$\eta = (s_0, \mu_0) \xrightarrow{e_1, t_1} (s_1, \mu_1) \xrightarrow{e_2, t_2} \ldots \xrightarrow{e_n, t_n} (s_n, \mu_n)$$

where $s_i \in S, e_i \in \Delta$ for all $i \geq 1$ and $\mu_i$ is a clock valuation for each $i$ such that the following hold

- $\mu_0(x) = 0$ for all $x \in X$
- For each $i \in \{1, 2, \ldots, n\}$, $e_i = (s_{i-1}, \sigma_i, \Psi_i, \gamma_i, s_i) \in \Delta$ for some $\sigma_i, \Psi_i$, and $\gamma_i$.
- For $i \geq 1$, $\mu_i(x) = \mu_{i-1}(x) + (t_i - t_{i-1})$ if $x \notin \gamma_i$ and $\mu_{i+1}(x) = 0$ if $x \in \gamma_i$.
- Finally, $\mu_{i-1} + (t_i - t_{i-1}) \models \psi$ for every $\psi \in \Psi_i$ ($t_0$ is defined to be 0).

The execution is said to be accepting if $s_n \in F$. The duration of the execution, denoted $|\eta|$, is defined to be $t_n$. Define $\mathsf{exec}^{pt}(\mathcal{B})$ to be the set of all *accepting* executions of $\mathcal{B}$. For an execution $\eta$, and $t \in \mathbb{R}_{\geq 0}$, we will say $t \in \eta$ iff $t = t_i$ for some $i$. For $t \in \eta$, where $t = t_i$ define $\sigma_\eta(t) = \sigma_i$. For $t \leq |\eta|$, let $t_i = \sup\{x \in \eta \,|\, x \leq t\}$. We will say an execution

$$\eta' = (s_0, \mu_0) \xrightarrow{e_1, t_1} (s_1, \mu_1) \xrightarrow{e_2, t_2} \ldots \xrightarrow{e_i, t_i} (s_i, \mu_i) \xrightarrow{e'_{i+1}, t'_{i+1}} (s'_{i+1}, \mu'_{i+1}) \xrightarrow{e'_{i+1}, t'_{i+1}} \ldots \xrightarrow{e'_m, t'_m} (s_m, \mu_m)$$

with $t'_{i+1} > t$ is an extension of $\eta$ from $t$.

### B. HCMTL* in Point-Based Semantics

Given an HCMTL* formula $\varphi$ with free path variables $\mathcal{V}$ and a timed automaton in the point-based semantics $\mathcal{B}$, a path environment is a map $\Gamma : \mathcal{V} \to \mathsf{exec}^{pt}(\mathcal{B})$. We will say $t \in \mathbb{R}_{\geq 0}$ is an event point in $\Gamma$ if $t \in \Gamma(\pi)$ for some $\pi \in \mathcal{V}$, and we denote this by $t \in \Gamma$. For $t \in \mathbb{R}_{\geq 0}$, and $\dagger$ taking values in $\mathcal{V} \cup \{\epsilon\}$, the satisfaction relation $\Gamma, t, \dagger \,{}^{pt}\!\models_{\mathcal{B}} \varphi$ is defined inductively as follows:

- $\Gamma, t, \dagger \,{}^{pt}\!\models_{\mathcal{B}} p_\pi$ iff $t \in \Gamma(\pi)$ and $p \in \sigma_{\Gamma(\pi)}(t)$.
- $\Gamma, t, \dagger \,{}^{pt}\!\models_{\mathcal{B}} \neg p_\pi$ iff $t \in \Gamma(\pi)$ and $p \notin \sigma_{\Gamma(\pi)}(t)$.
- $\Gamma, t, \dagger \,{}^{pt}\!\models_{\mathcal{B}} \varphi_1 \vee \varphi_2$ iff $\Gamma, t, \dagger \,{}^{pt}\!\models_{\mathcal{B}} \varphi_1$ or $\Gamma, t, \dagger \,{}^{pt}\!\models_{\mathcal{B}} \varphi_2$.
- $\Gamma, t, \dagger \,{}^{pt}\!\models_{\mathcal{B}} \varphi_1 \wedge \varphi_2$ iff $\Gamma, t, \dagger \,{}^{pt}\!\models_{\mathcal{B}} \varphi_1$ and $\Gamma, t, \dagger \,{}^{pt}\!\models_{\mathcal{B}} \varphi_2$.
- $\Gamma, t, \dagger \,{}^{pt}\!\models_{\mathcal{B}} F_I \varphi$ iff there exists $t' > t$ such that $t' - t \in I$, $t' \in \Gamma$ and $\Gamma, t', \dagger \,{}^{pt}\!\models_{\mathcal{B}} \varphi$.
- $\Gamma, t, \dagger \,{}^{pt}\!\models_{\mathcal{B}} G_I \varphi$ iff for all $t' > t$ such that $t' - t \in I$ and $t' \in \Gamma$, $\Gamma, t', \dagger \,{}^{pt}\!\models_{\mathcal{B}} \varphi$.
- $\Gamma, t, \dagger \,{}^{pt}\!\models_{\mathcal{B}} \varphi_1 U_I \varphi_2$ iff there exists $t' > t$ such that $t' - t \in I$, $t' \in \Gamma$, $\Gamma, t', \dagger \,{}^{pt}\!\models_{\mathcal{B}} \varphi_2$; and $\Gamma, t'', \dagger \,{}^{pt}\!\models_{\mathcal{B}} \varphi_1$ for all $t < t'' < t'$ such that $t'' \in \Gamma$.
- $\Gamma, t, \dagger \,{}^{pt}\!\models_{\mathcal{B}} \exists \pi \varphi$ iff there is an execution $\eta \in \mathsf{exec}^{pt}(\mathcal{B})$ such that $\Gamma[\pi \mapsto \eta], t, \pi \,{}^{pt}\!\models_{\mathcal{B}} \varphi$ and either (a) $\dagger = \epsilon$ and $t = 0$, or (b) $t \leq |\Gamma(\dagger)|$ and $\eta$ is an extension of $\Gamma(\dagger)$ from $t$.
- $\Gamma, t, \dagger \,{}^{pt}\!\models_{\mathcal{B}} \forall \pi \varphi$ iff for every $\eta \in \mathsf{exec}^{pt}(\mathcal{B})$, if either (a) $\dagger = \epsilon$ and $t = 0$, or (b) $t \leq |\Gamma(\dagger)|$ and $\eta$ is an extension of $\Gamma(\dagger)$ from $t$, then $\Gamma[\pi \mapsto \eta], t, \pi \,{}^{pt}\!\models_{\mathcal{B}} \varphi$.

We can define time bounded semantics for a time bound $N \in \mathbb{R}_{>0}$ in a manner similar as we did for the interval-based semantics in Section II-C.

### C. Point-Based vs Interval-Based Semantics

The first observation we make between the two semantics is that timed automata in the interval-based semantics can simulate timed automata in the point-based semantics. The idea is that an execution of the automaton $\mathcal{B}$

$$\eta = (s_0, \mu_0) \xrightarrow{e_1,t_1} (s_1, \mu_1) \xrightarrow{e_2,t_2} \dots \xrightarrow{e_n,t_n} (s_n, \mu_n)$$

can be thought of in the interval semantics as the execution

$$\begin{aligned}\rho =&(s_0, [0, t_1)), (e_1, [t_1, t_1]), (s_1, (t_1, t_2)), (e_2, [t_2, t_2]),\\ &\dots, (s_{n-1}, (t_{n-1}, t_n), (e_n, [t_n, t_n]).\end{aligned}$$

where transitions are marked by singular intervals. Here, if $t_1 = 0$, the execution starts at $(e_1, [t_1, t_1])$. To achieve this, we construct an automaton in the interval-based semantics, we will call it $\mathcal{B}_{\mathsf{ib}}$, that has as its states all the states and edges of $\mathcal{B}$. $\mathcal{B}_{\mathsf{ib}}$ alternates between states that correspond to states of $\mathcal{B}$ and states that correspond to edges of $\mathcal{B}$. To ensure that this automaton remains in the edge states only for singular intervals, we have a special clock $x_{sing}$. This clock gets reset whenever we enter a state corresponding to an edge, and that state has the constraint $x_{sing} = 0$. In $\mathcal{B}_{\mathsf{ib}}$, the states corresponding to edges have the clock constraints of the edge to ensure the timing constraints of $\eta$ are satisfied.

Given a timed automaton $\mathcal{B} = (2^P, S, s_0, X, \Delta, F)$ in the point-based semantics, define a timed automaton $\mathcal{B}_{\mathsf{ib}}$ over the set of propositions $P \cup S \cup \{\#\}$ in the interval-based semantics as, $\mathcal{B}_{\mathsf{ib}} = (V, V_0, \alpha, X', \beta, E, V_F)$ where:

- $V = S \cup \{e \mid e \in \Delta\}$.
- $V_0 = \{s_0\} \cup \{e \in \Delta \mid e = (s_0, \sigma, \psi, \gamma, s')\}$.
- $\alpha(s) = \{s\}$ for all $s \in S$ and $\alpha(e) = \sigma \cup \{\#\}$ for $e = (s_1, \sigma, \Psi, \gamma, s_2)$.
- $X' = X \cup \{x_{sing}\}$.
- $\beta(s, x) =$ true for all $s \in S$ and $x \in X'$. For $e = (s_1, \sigma, \Psi, \gamma, s_2)$, $\beta(e, x) = \wedge_{\psi \in G_x} \psi$ where $G_x = \Psi \cap \Phi(x)$ for $x \in X$, and $\beta(e, x_{sing}) = \{x_{sing} = 0\}$.
- $E = \{(s, e, \{\text{true}\}, \{x_{sing}\}) \mid e = (s, \sigma, \Psi, \gamma, s') \in \Delta\} \cup \{(e, s', \{\text{true}\}, \gamma) \mid e = (s, \sigma, \Psi, \gamma, s') \in \Delta\}$.
- $V_F = \{e \in \Delta \mid e = (s_1, \sigma, \Psi, \gamma, s_2)$ and $s_2 \in F\}$.

The $\#$ proposition is used to mark points where transitions occur and is used later to reduce the verification problem in the point-based semantics to the verification problem in the interval-based semantics. Consider the map $\chi : \mathsf{exec}^{pt}(\mathcal{B}) \to \mathsf{exec}(\mathcal{B}_{\mathsf{ib}})$ that maps an execution

$$\eta = (s_0, \mu_0) \xrightarrow{e_1,t_1} (s_1, \mu_1) \xrightarrow{e_2,t_2} \dots \xrightarrow{e_n,t_n} (s_n, \mu_n)$$

of $\mathcal{B}$ to the execution

$$\begin{aligned}\rho =&(s_0, [0, t_1)), (e_1, [t_1, t_1]), (s_1, (t_1, t_2)), (e_2, [t_2, t_2]),\\ &\dots, (s_{n-1}, (t_{n-1}, t_n), (e_n, [t_n, t_n])\end{aligned}$$

of $\mathcal{B}_{\mathsf{ib}}$. The automaton $\mathcal{B}_{\mathsf{ib}}$ simulates exactly the set of accepting executions of $\mathcal{B}$ when transformed using the map $\chi$. Thus, we have the following lemma.

**Lemma 12.** *$\chi$ is a bijection from $\mathsf{exec}^{pt}(\mathcal{B})$ to $\mathsf{exec}(\mathcal{B}_{\mathsf{ib}})$.*

Just as timed automata in the interval-based semantics can simulate timed automata in the point-based semantics, HCMTL* in the interval-based semantics is at least as expressive as HCMTL* in the point-based semantics. To show this, given an HCMTL* formula $\varphi$ in the point-based semantics, we will construct a formula $\varphi^{\mathsf{ib}}$ which when interpreted in the interval-based semantics, expresses exactly the set of all path environments that satisfy $\varphi$ under a suitable encoding of path environments. Given a timed automaton $\mathcal{B}$ in the point-based semantics, and a path environment $\Gamma : \mathcal{V} \to \mathsf{exec}^{pt}(\mathcal{B})$, we can lift the map $\chi$ to $\Gamma$ to get a path environment $\Pi_\Gamma : \mathcal{V} \to \mathsf{exec}(\mathcal{B}_{\mathsf{ib}})$ defined as $\Pi_\Gamma(\pi) = \chi(\Gamma(\pi))$. Similarly, for any path environment $\Pi : \mathcal{V} \to \mathsf{exec}(\mathcal{B}_{\mathsf{ib}})$, we get a path environment $\Gamma_\Pi : \mathcal{V} \to \mathsf{exec}^{pt}(\mathcal{B})$ by lifting the map $\chi^{-1}$. We have the following expressiveness result:

**Lemma 13.** *Given an HCMTL* formula $\varphi$ in the point-based semantics over a set of propositions $P$ with free path variables $\mathcal{V}$, there is a formula $\varphi^{\mathsf{ib}}$ in the interval-based semantics over $P \cup \{\#\}$ such that for any timed automaton $\mathcal{B}$ in the point-based semantics and path environment $\Gamma : \mathcal{V} \to \mathsf{exec}^{pt}(\mathcal{B})$, if $\Gamma, t, \dagger \models^{pt}_{\mathcal{B}} \varphi$ then $\Pi_\Gamma, t, \dagger \models_{\mathcal{B}_{\mathsf{ib}}} \varphi^{\mathsf{ib}}$. Conversely, for any path environment $\Pi : \mathcal{V} \to \mathsf{exec}(\mathcal{B}_{\mathsf{ib}})$, if $\Pi, t, \dagger \models_{\mathcal{B}_{\mathsf{ib}}} \varphi^{\mathsf{ib}}$ then $\Gamma_\Pi, t, \dagger \models^{pt}_{\mathcal{B}} \varphi$.*

We present the construction of $\varphi^{\mathsf{ib}}$. $\varphi^{\mathsf{ib}}$ is defined inductively as follows:

- $\varphi = p_\pi$. In the point-based semantics, this corresponds to $p$ being true at some event in execution $\pi$ at time $t$. The interval-based semantics doesn't have a notion of an event occurring at time $t$ since the system is always observed. To capture the point-based semantics, we use the proposition $\#$ which is true only when some transition/event occurs in $\mathcal{B}$. We define $\varphi^{\mathsf{ib}} = \#_\pi \wedge p_\pi$.
- $\varphi = \neg p_\pi$. As above, we define $\varphi^{\mathsf{ib}} = \#_\pi \wedge \neg p_\pi$.

Inductive Cases

- Conjunction and disjunctions have the same semantics, so we just have
$$(\varphi_1 \vee \varphi_2)^{\mathsf{ib}} = {\varphi_1}^{\mathsf{ib}} \vee {\varphi_2}^{\mathsf{ib}}$$
$$(\varphi_1 \wedge \varphi_2)^{\mathsf{ib}} = {\varphi_1}^{\mathsf{ib}} \wedge {\varphi_2}^{\mathsf{ib}}$$
- $\varphi = F_I \varphi_1$. In the point-based semantics this says that at some time $t' \in t + I$, such that an event occurs at $t'$, $\varphi_1$ is true. In the interval-based semantics, we again use the $\#$ proposition to check the truth of $\varphi_1$ only at time points where event occurs. We define $\varphi^{\mathsf{ib}}$ as:
$$\varphi^{\mathsf{ib}} = F_I(\vee_{\pi \in \mathcal{V}} \#_\pi \wedge {\varphi_1}^{\mathsf{ib}}).$$
- $\varphi = G_I \varphi_1$. Similar to the case of $F_I$, we define $\varphi^{\mathsf{ib}}$ as:
$$\varphi^{\mathsf{ib}} = G_I((\vee_{\pi \in \mathcal{V}} \#_\pi) \implies {\varphi_1}^{\mathsf{ib}}).$$

- $\varphi = \varphi_1 U_I \varphi_2$. Using the same idea as $F_I$ we have
$$\varphi^{\mathsf{ib}} = (\vee_{\pi \in \mathcal{V}} \#_\pi \implies {\varphi_1}^{\mathsf{ib}}) U_I (\vee_{\pi \in \mathcal{V}} \#_\pi \wedge {\varphi_2}^{\mathsf{ib}})$$
- Quantification. The semantics of existential and universal quantification is identical in both point-based and interval-based semantics. Hence, we have,
$$(\exists \pi\, \varphi_1)^{\mathsf{ib}} = \exists \pi\, {\varphi_1}^{\mathsf{ib}}$$
$$(\forall \pi\, \varphi_1)^{\mathsf{ib}} = \forall \pi\, {\varphi_1}^{\mathsf{ib}}$$

This translation works even if we consider the bounded time semantics by restricting all executions to a time bound $N \in \mathbb{R}_{>0}$. As a corollary of this, we get

**Corollary 14.** *For any HCMTL* formula $\varphi$ and a timed automaton $\mathcal{B}$ in the point-wise semantics,*
$$\{\}, 0, \epsilon \models^{pt}_{\mathcal{B}} \varphi \iff \{\}, 0, \epsilon \models_{\mathcal{B}_{\mathsf{ib}}} \varphi^{\mathsf{ib}}.$$

Hence, the interval-based semantics of HCMTL* is at least as expressive as the point-based semantics. Thus, bounded time verification problem in point-based semantics reduces to bounded time verification problem in the interval-based semantics which gives us the following result.

**Theorem 15.** *Bounded time verification problem for HCMTL* in the point-based semantics is decidable.*

It was shown by Hsi-Ming Ho et. al. [23] that model checking HyperMITL in the point-based semantics is undecidable by reducing the universality problem of timed automata to model checking HyperMITL, which is known to be undecidable. In a similar manner, universality problem for timed automata can be reduced to the general verification problem of the HyperMITL fragment of HCMTL* in the point-based semantics. Thus, HyperMITL and HCMTL* are both undecidable in the point-based semantics. By our corollary 14, the general verification problem for HyperMITL and HCMTL* is undecidable in the interval-based semantics also. Hence, we have the following result which also implies Theorem 3:

**Theorem 16.** *The general verification problem for HCMTL* is undecidable in both the point-based and interval-based semantics. In fact, the verification problem is undecidable even for the fragment* HyperMITL.

## VI. Related Work

After Clarkson and Schneider introduced hyperproperties, there has been an increasing interest in verifying hyperproperties. Clarkson et al. proposed temporal logic HyperLTL and HyperCTL* to describe hyperproperties and showed that when restricted to finite Kripke structures, the model-checking problem of HyperLTL and HyperCTL* is decidable. They also establish complexity results for the verification problem. Automated tools like MCHyper [19], AutoHyper [8] for model checking and satisfiability checking, EAHyper [17] for satisfiability checking and RVHyper [18] for runtime monitoring have been built.

To express security hyperproperties in a timed setting, different hyper-timed logics have been proposed. A pioneering work is [26], which extends STL with quantification over real-time signals and is studied over cyber-physical systems.

Linear-time HyperMTL that extends MTL was introduced in [9]. The real-time systems in [9] are modeled as timed Kripke structures, that are Kripke structures with time elapsing on transitions. The semantics used in [9] is point-based, and the model of time is discrete-time. The logic is defined over finite timed words but is syntactically different from ours. They show that the verification problem for the logic is decidable for a nontrivial fragment of the logic by reducing the problem to checking untimed hyperproperties.

In [10], linear-time temporal logic Time Window Temporal Logic (TWTL) is extended to reason about hyperproperties. Like [9], the models are timed Kripke structures, the semantics considered is point-based, and the model of time is discrete time. The verification problem for the resulting logic is shown to be undecidable, and the model-checking algorithms are given for the alternation-free fragment of the logic.

Linear-time HyperMITL has been proposed in [23] by extending MITL. Unlike our work, they also consider past operators in the logic. The semantics used in [23] is point-based, and the model of time used is continuous/dense. [23] consider decidability for many fragments of HyperMITL, and most of the fragments are undecidable over the unbounded time domain. The most interesting decidability result is verifying HyperMITL is decidable for bounded time domains. While the proof is not provided, the authors hint that the proof could be accomplished by a reduction to the satisfiability problem for QPTL [33] which is known to be decidable [33].

Our work extends MTL to HCMTL* that allows us to express branching hyperproperties in addition to linear-time hyperproperties. Like [23], the model of time is taken to be dense. However, we also consider the interval-based semantics in addition to point-based semantics. We show that the time-bounded verification for HCMTL* is decidable. In contrast to [23], our decidability result is obtained by a reduction to the satisfiability problem of MSO with order and successor. The challenge is to encode the semantics of branching logic as well as time automaton into an MSO formula. We also show that the decision problems for verifying HCMTL* under point-based semantics can be reduced to the verification problem for interval-based semantics. This allows us to transfer decidability results under interval-based semantics to point-based semantics, and undecidability results from point-based semantics to interval-based semantics. One difference from [9], [23] is that they distinguish between synchronous and asynchronous hyperproperties. Intuitively, in synchronous semantics, observations on all traces happen at the same time, while in asynchronous semantics they may happen at different times. Since our focus is interval-based semantics, our semantics is asynchronous in principle.

HyperTidy CTL* logic is presented in [7] to reason about timed hyperproperties of timed cryptographic protocols in the Dolev-Yao model [15]. HyperTidy CTL* is also a branching logic and is similar to HCMTL*, but has significant differences. First, it is interpreted over timed processes and not

timed automata, a variant of applied pi calculus [2] augmented with timing constructs. The logic has special atomic constructors and atomic formulas to model the attacker's knowledge and actions. For example, it has the atomic formula $X \vdash u$, where $X$ is a variable that ranges over *recipes*. Intuitively, a *recipe* is a term in applied pi calculus [2] that represents a computation by a Dolev-Yao attacker from the messages it possesses, and the formula $X \vdash u$ represents that the message $u$ can be computed from the recipe assigned to $X$. The logic allows quantification over recipe and message variables in addition to the path variables. Further, the temporal operators are not annotated explicitly with intervals, and the semantics is equivalent to taking all intervals to be $(0, \infty)$. It is shown in [7] that the problem of checking that a process satisfies a HyperTidy CTL* is undecidable, even for the LTL fragment. The proof relies on the fact that the problem of checking whether the attacker can compute a message $u$ from a finite set of messages is undecidable [1]. This undecidability result is incomparable to the undecidability of HCMTL* presented in this paper as both the models and the logic in [7] are much richer than ours.

In [31], a variant of HyperTidy CTL* is considered. The formula $X \vdash u$ is replaced by $K(u)$, which intuitively means that $u$ can be computed by the attacker from its intercepted messages using some recipe. Thus, there are no recipe variables. The quantification of message variables also takes a restricted form. The temporal operators are now decorated with intervals as they are needed to specify liveness properties such as timeliness and fairness properties as discussed in Section III. They consider the problem of verifying whether a process satisfies a HyperTidy CTL* formula is satisfied by a process when the number of protocol sessions is bounded and when the cryptographic primitives are modeled using subterm-convergent equational theories [1]. They show that the verification problem becomes **EXPSPACE** complete for this variant. The decision procedure is based on constraint solving. Please note that this decision procedure is incomparable to ours as we consider abstract finite timed automata and not processes. Further, the assumption of a bounded number of sessions means that the "transition system" underlying a process in [31] is acyclic, and any trace necessarily has a bounded number of actions/observations that depend on the process being verified. We make no such assumptions, and traces can have any number of observations. However, the transition system underlying [31] is infinite branching; hence, the total number of traces is still infinite in this setting. In contrast, our transition systems are finite-branching, and the number of traces is infinite because we allow for any number of actions along a trace.

## VII. CONCLUSIONS AND FUTURE WORK

We introduce an extension of MTL that expresses branching hyperproperties of real time systems. We investigate the verification problems associated with this logic against timed automata for both interval-based and point-based semantics. We show that the problem is undecidable for both semantics when the time domain is unbounded. However, when bounded time domains are considered, the verification problem becomes decidable for both semantics.

**Complexity.** The decidability result is established by reducing the problem to checking the satisfaction problem of MSO with ordering and successor over bounded time domains. While this reduction establishes decidability, the complexity of deciding MSO$(<, +1)$ over bounded time is non-elementary [28]. Hence, the decision procedure presented in this paper has non-elementary complexity. By non-elementary, we mean that the runtime cannot be bounded by a tower of exponentials whose height is independent of the automaton's size and the formula's size. Thus, our analysis might not provide a tight complexity bound. Given that the (space) complexity of verifying HyperLTL and HyperCTL* for untimed systems is a tower of exponentials whose height is the alternation depth [12], [19], it is unlikely that HCMTL* will have elementary complexity over bounded time horizon.

Thus, we plan to explore the complexity of verification in terms of alternation depth. One potential method of obtaining a tight complexity bound is first studying the complexity of verification over rational time (which is still a dense time domain). Over bounded rational time, MSO$(<, +1)$ can be encoded into $S2S$, the monadic second order logic of two successors interpreted over binary trees. Hence, the bounded verification problem of HCMTL* over rational time can be reduced to a satisfiability problem over $S2S$. Satisfiability of $S2S$ has a decision procedure via a reduction to the emptiness problem of alternating tree automata. This procedure has the complexity of a tower of exponentials having a height in the order of alternation depth in the MSO$(<, +1)$ formula. This implies that the complexity of bounded-time verification of HCMTL* when the model of dense time is the set of rationals is a tower of exponentials having height linear in alternation depth in the HCMTL* formula. Please note that this is an upper bound on the complexity of the verification problem, and we plan to investigate the exact characterization of the complexity of verifying HCMTL* over a bounded time domain in the future.

**Algorithmic implications.** We plan to implement the decision procedure presented in this paper using existing tools that implement decision procedures on monadic second-order logic, such as the MONA tool [20].


## ACKNOWLEDGEMENTS

Rohit Chadha was partially supported by NSF CCF grant 1900924, and Nabarun Deka, Minjian Zhang and Mahesh Viswanathan were partially supported by NSF SHF 1901069 and NSF CCF 2007428. We thank the anonymous reviewers for carefully reviewing our paper and providing insightful feedback and comments.


## REFERENCES


[1] M. Abadi and V. Cortier, "Deciding knowledge in security protocols under equational theories," *Theoretical Computer Science*, vol. 387, no. 1-2, pp. 2–32, 2006.

[2] M. Abadi and C. Fournet, "Mobile values, new names, and secure communication," in *28th ACM Symp. on Principles of Programming Languages (POPL'01)*, 2001, pp. 104–115.

[3] R. Alur and T. Henzinger, "Real-time logics: Complexity and expressiveness," *Information and Computation*, vol. 104, no. 1, pp. 35–77, 1993. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0890540183710254

[4] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0304397594900108

[5] R. Alur, T. Feder, and T. A. Henzinger, "The benefits of relaxing punctuality," *J. ACM*, vol. 43, no. 1, p. 116–146, jan 1996. [Online]. Available: https://doi.org/10.1145/227595.227602

[6] R. Alur and T. A. Henzinger, "A really temporal logic," *J. ACM*, vol. 41, no. 1, p. 181–203, jan 1994. [Online]. Available: https://doi.org/10.1145/174644.174651

[7] G. Barthe, U. Dal Lago, G. Malavolta, and I. Rakotonirina, "Tidy: Symbolic verification of timed cryptographic protocols," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 263–276. [Online]. Available: https://doi.org/10.1145/3548606.3559343

[8] R. Beutner and B. Finkbeiner, "Autohyper: Explicit-state model checking for hyperltl," in *Tools and Algorithms for the Construction and Analysis of Systems*, S. Sankaranarayanan and N. Sharygina, Eds. Cham: Springer Nature Switzerland, 2023, pp. 145–163.

[9] B. Bonakdarpour, P. Prabhakar, and C. Sánchez, "Model checking timed hyperproperties in discrete-time systems," in *NASA Formal Methods*, R. Lee, S. Jha, A. Mavridou, and D. Giannakopoulou, Eds. Cham: Springer International Publishing, 2020, pp. 311–328.

[10] E. Bonnah, L. Nguyen, and K. A. Hoque, "Model checking time window temporal logic for hyperproperties," in *Proceedings of the 21st ACM-IEEE International Conference on Formal Methods and Models for System Design*, ser. MEMOCODE '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 100–110. [Online]. Available: https://doi.org/10.1145/3610579.3611077

[11] M. Brusó, K. Chatzikokolakis, S. Etalle, and J. den Hartog, "Linking unlinkability," in *Trustworthy Global Computing*, C. Palamidessi and M. D. Ryan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 129–144.

[12] M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez, "Temporal logics for hyperproperties," in *Principles of Security and Trust - Third International Conference*. Springer, 2014, pp. 265–284.

[13] M. R. Clarkson and F. B. Schneider, "Hyperproperties," in *Proceedings of the 21st IEEE Computer Security Foundations Symposium*. IEEE Computer Society, 2008, pp. 51–65.

[14] N. Deka, M. Zhang, R. Chadha, and M. Viswanathan, "Deciding branching hyperproperties for real time systems," *arXiv*, 2024.

[15] D. Dolev and A. Yao, "On the security of public key protocols," in *Proc. of the 22nd Symp. on Foundations of Computer Science*. IEEE Comp. Soc. Press, 1981, pp. 350–357.

[16] E. A. Emerson and J. Y. Halpern, ""Sometimes" and "Not Never" revisited: On branching versus linear time," in *Conference Record of the Tenth Annual ACM Symposium on Principles of Programming Languages*. ACM Press, 1983, pp. 127–140.

[17] B. Finkbeiner, C. Hahn, and M. Stenger, "Eahyper: Satisfiability, implication, and equivalence checking of hyperproperties," in *Computer Aided Verification*, R. Majumdar and V. Kunčak, Eds. Cham: Springer International Publishing, 2017, pp. 564–570.

[18] B. Finkbeiner, C. Hahn, M. Stenger, and L. Tentrup, "RVHyper: A Runtime Verification Tool for Temporal Hyperproperties," in *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2018, pp. 194–200.

[19] B. Finkbeiner, M. N. Rabe, and C. Sánchez, "Algorithms for model checking HyperLTL and HyperCTL*," in *Computer Aided Verification - 27th International Conference*. Springer, 2015, pp. 30–48.

[20] J. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm, "Mona: Monadic second-order logic in practice," in *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS '95, LNCS 1019*, 1995.

[21] T. A. Henzinger, J.-F. Raskin, and P.-Y. Schobbens, "The regular real-time languages," in *Automata, Languages and Programming*, K. G. Larsen, S. Skyum, and G. Winskel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 580–591.

[22] T. A. Henzinger, "The temporal specification and verification of real-time systems," Ph.D. dissertation, Stanford, CA, USA, 1992, uMI Order No. GAX92-06781.

[23] H.-M. Ho, R. Zhou, and T. M. Jones, "Timed hyperproperties," *Information and Computation*, vol. 280, p. 104639, 2021.

[24] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-time Systems*, vol. 2, no. 4, pp. 255–299, 1990.

[25] J. McLean, "Proving noninterference and functional correctness using traces," *Journal of Computer Security*, vol. 1, no. 1, pp. 37–58, 1992.

[26] L. V. Nguyen, J. Kapinski, X. Jin, J. V. Deshmukh, and T. T. Johnson, "Hyperproperties of real-valued signals," ser. MEMOCODE '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 104–113. [Online]. Available: https://doi.org/10.1145/3127041.3127058

[27] J. Ouaknine and J. Worrell, "On the decidability of metric temporal logic," in *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*, 2005, pp. 188–197.

[28] J. Ouaknine, A. Rabinovich, and J. Worrell, "Time-bounded verification," in *CONCUR 2009 - Concurrency Theory*, M. Bravetti and G. Zavattaro, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 496–510.

[29] J. Ouaknine and J. Worrell, "Towards a theory of time-bounded verification," in *Automata, Languages and Programming*, S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. G. Spirakis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 22–37.

[30] A. Pnueli, "The temporal logic of programs," in *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1977, pp. 46–57.

[31] I. Rakotonirina, G. Barthe, and C. Schneidewind, "Decision and complexity of Dolev-Yao hyperproperties," *Proc. ACM Program. Lang.*, vol. 8, no. POPL, jan 2024. [Online]. Available: https://doi.org/10.1145/3632906

[32] J.-F. Raskin and P.-Y. Schobbens, "State clock logic: A decidable real-time logic," in *Hybrid and Real-Time Systems*, O. Maler, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 33–47.

[33] A. P. Sistla, M. Y. Vardi, and P. Wolper, "The complementation problem for büchi automata with appplications to temporal logic," *Theoretical Computer Science*, vol. 49, pp. 217–237, 1987.