Model Checking Time Window Temporal Logic for Hyperproperties

Ernest Bonnah
University of Missouri-Columbia
Columbia, Missouri, USA
ernest.bonnah@mail.missouri.edu

Luan Viet Nguyen
University of Dayton
Dayton, Ohio, USA
lnguyen1@udayton.edu

Khaza Anuarul Hoque
University of Missouri-Columbia
Columbia, Missouri, USA
hoquek@umsystem.edu

ABSTRACT

Hyperproperties extend trace properties to express properties of sets of traces, and they are increasingly popular in specifying various security and performance-related properties in domains such as cyber-physical systems, smart grids, and automotive. This paper introduces HyperTWTL, which extends Time Window Temporal Logic (TWTL)–a domain-specific formal specification language for robotics, by allowing explicit and simultaneous quantification over multiple execution traces. We propose two different semantics for HyperTWTL, *synchronous* and *asynchronous*, based on the alignment of the timestamps in the traces. Consequently, we demonstrate the application of HyperTWTL in formalizing important information-flow security policies and concurrency for robotics applications. Furthermore, we introduce a model checking algorithm for verifying fragments of HyperTWTL by reducing the problem to a TWTL model checking problem.

CCS CONCEPTS

• **Theory of computation → Modal and temporal logics**; **Verification by model checking**; **Logic and verification**.

KEYWORDS

Hyperproperties, Model Checking, Time Window Temporal Logic, Robotics

ACM Reference Format:
Ernest Bonnah, Luan Viet Nguyen, and Khaza Anuarul Hoque. 2023. Model Checking Time Window Temporal Logic for Hyperproperties. In *21st ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE '23), September 21–22, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3610579.3611077

1 INTRODUCTION

Hyperproperties [21] extend the notion of trace properties [4] from a set of traces to a set of sets of traces. In other words, a hyperproperty specifies system-wide properties in contrast to the property of just individual traces. This allows us to specify a wide range of properties related to information-flow security [29, 49], consistency models in concurrent computing [10, 24], robustness models in cyber-physical systems [12, 28], and also service level agreements (SLA) [21]. Motivated by the expressiveness of hyperproperties, several hyper-temporal logics, such as HyperLTL [20], HyperSTL [38], and HyperMTL [11], were recently proposed by extending the conventional temporal logics such as Linear Temporal Logic (LTL) [41], Signal Temporal Logic (STL) [37], and Metric Temporal Logic (MTL) [33], respectively. Consequently, various model checking techniques have been proposed for verifying HyperLTL [20, 22, 25, 27], HyperMTL [11, 30], HyperMITL [31], HyperSTL [38] specifications employing alternating automata, model checking, strategy synthesis, and several other methods [32].

Time bounded specifications are common in many applications, such as robotics. Such specifications with time constraints cannot be expressed using LTL, however they can be expressed using Metric Temporal Logic (MTL) [33], Signal Temporal Logic (STL)[37], Bounded Linear Temporal Logic (BLTL)[46] and Time Window Temporal Logic (TWTL) [48]. The TWTL has rich semantics and is known for expressing specification more compactly when compared to Metric Temporal Logic (MTL) [33], Signal Temporal Logic (STL)[37], Bounded Linear Temporal Logic (BLTL)[46]. For instance, let us consider a specification as "*stay at Q for the first 5-time steps within the time window [0, 10]*". This can be expressed in TWTL as $[\mathbf{H}^5 Q]^{[0,10]}$. The exact specification can be expressed in STL as $\mathbf{F}_{[0,10-5]}\mathbf{G}_{[0,5]}Q$ where the outermost time window needs to be modified with respect to the inner time window. Furthermore, TWTL presents an explicit concatenation operator ($\odot$), which is very useful in expressing serial tasks in robotic mission specification and planning. Thus, TWTL is becoming popular in different application domains [2, 5, 6, 13, 16, 40]. However, the conventional TWTL can only express trace properties (can reason about individual traces). This limits their application to evaluating a wide range of security properties as evaluating them requires reasoning about multiple traces. A straightforward extension of TWTL to a hyper-temporal logic will result in a flawed formalism and impractical for robotic applications. There are multiple challenges in designing a domain-specific temporal logic for timed hyperproperties. First, the alignment of time stamps across multiple traces of a system must be taken into consideration. This is due to the complexity of interpreting time stamps that are not aligned across traces. Additionally, the framework of any hyper-temporal logic has to consider the speed of time with which time-stamped traces proceed. This is because

Table 1: The representation of φ in HyperTWTL, HyperSTL, HyperMTL

HyperTWTL	$\forall\pi\forall\pi' \cdot [\mathbf{H}^5\, A_\pi]^{[0,6]} \odot [\mathbf{H}^2\, B_{\pi'}]^{[7,10]}$
HyperSTL	$\forall\pi\forall\pi' \cdot (\mathbf{F}_{[0,6-5]}\mathbf{G}_{[0,5]}A_\pi) \wedge (\mathbf{F}_{[7,10-2]}\mathbf{G}_{[0,2]}B_{\pi'})$
HyperMTL	$\forall\pi\forall\pi' \cdot \bigvee_{i=0}^{6-5}(\mathbf{G}_{[i,i+5]}A_\pi \wedge \bigvee_{j=i+5+7}^{i+5+10-2}\mathbf{G}_{[j,j+2]}B_{\pi'})$

traces may not proceed at the same speed of time. To reason about traces that move at different speeds of time, *asynchronous semantics* need to be allowed.

A hyper-temporal logic, HyperTWTL, extends the classical semantics of TWTL with trace variables and explicit existential and universal quantifiers over multiple execution traces. HyperTWTL can express bounded hyperproperties more compactly when compared to HyperMTL and HyperSTL. For instance, consider a hyperproperty that requires that "*for any pair of traces π and π' of a system, A should hold for 5-time steps in trace π within the time bound $[0, 6]$, afterwards B should also hold for 2-time steps in trace π' within the time bound $[7, 10]$*." This requirement φ can be expressed using HyperTWTL, HyperSTL, and HyperMTL formalisms as shown in Table 1. The compact syntax of HyperTWTL allows for a more succinct representation of this requirement than HyperMTL and HyperSTL, which require nested operators, shifted time windows, and the disjunction of several sub-formulae. For instance, in Table 1, it can also be observed that the given requirement can be formalized in HyperTWTL formula with total 5 temporal operators (without considering the quantifiers). This same requirement can be formalized as a HyperMTL formula using 11 temporal operators (excluding the quantifiers). This example shows the succinct characteristics of HyperTWTL over HyperMTL. Indeed, with the increasing complexity of requirements, the complexity of HyperMTL formulae will also grow, which makes the formal analysis of HyperMTL formulae very expensive for complicated robotic applications.

In this paper, we describe a full version of HyperTWTL with two semantics, *synchronous* and *asynchronous*, to represent system behaviors as a sequence of events that can occur either at the same or different time points respectively. In the synchronous semantics, the operators are evaluated time-point by time-point similar to HyperLTL. In contrast, asynchronous semantics allows for evaluation over traces that proceed at different speed of time. We demonstrate how HyperTWTL can be used to formalize important security policies such as non-interference, opacity, countermeasures to side-channel timing attacks, and also concurrency-related properties, such as linearizability. Such security requirement are common not only in robotics but also in other computing domains at hardware, software and system level. Finally, we propose a model checking algorithm for fragments (alternation-free and k-alternations) of HyperTWTL by reducing the model checking problem to the TWTL model checking problem.

The rest of the paper is organized as follows: Section 2 reviews the preliminary concepts. The syntax and semantics of HyperTWTL are presented in Section 3. In Section 4, some applications of HyperTWTL are discussed. Our model checking algorithm for HyperTWTL is presented in section 5. We evaluate the feasibility of our proposed algorithm in Section 4. Related works are discussed in Section 7. Finally, Section 8 concludes the paper.

2 PRELIMINARIES

Let AP be a finite set of *atomic propositions* and $\Sigma = 2^{AP}$ be the powerset of AP. Let $\mathcal{A} = \mathbb{Z}_{\geq 0} \times \Sigma$ be the *alphabet*, where $\mathbb{Z}_{\geq 0}$ is the set of non-negative integers. A (time-stamped) *event* is a member of the alphabet $\mathcal{A}$ and is of the form (τ, e), where $\tau \in \mathbb{Z}_{\geq 0}$ and $e \in \Sigma$. A trace $t \in \mathcal{A}^\omega$ denotes an infinite sequence of events over $\mathcal{A}$, and $t \in \mathcal{A}^*$ denotes a finite sequence of events over $\mathcal{A}$. For a trace t, we denote $t[n].e$ as the event at time n, i.e. e_n and by $t[n].\tau$, we mean τ_n. Let $t[i, j]$ denote the subtrace of trace t starting from time i up to time j. We model timed systems as Timed Kripke Structures with the assigned time elapsed on the transitions.

Definition 1. A timed Kripke structure (TKS) is a tuple $\mathcal{T} = (S, S_{init}, \delta, AP, L)$ where

- S is a finite set of states;
- $S_{init} \subseteq S$ is the set of initial states;
- $\delta \subseteq S \times \mathbb{Z}_{\geq 0} \times S$ is a set of transitions;
- AP is a finite set of atomic propositions; and
- $L : S \to \Sigma$ is a labelling function on the states of $\mathcal{T}$

We require that for each $s \in S$, there exists a successor that can be reached in a finite number of transitions. Hence, all nodes without any outgoing transitions are equipped with self-loops such that $(s, 1, s) \in \delta$. A path over an TKS is an infinite sequence of states $s_0 s_1 s_2 \cdots \in S^\omega$, where $s_0 \in S_{init}$ and $(s_i, d_i, S_{i+1}) \in \delta$, for each $i \geq 0$. A trace over TKS is of the form: $t = (\tau_0, e_0)(\tau_1, e_1)(\tau_2, e_2) \ldots$, such that there exists a path $s_0 d_0 s_1 d_1 s_2 d_2 \cdots \in S^\omega$.

TWTL Syntax and Semantics. The set of TWTL formulae over a finite set of atomic propositions is defined by the following syntax:

$$\phi := \top \mid \mathbf{H}^d a \mid \mathbf{H}^d \neg a \mid \phi_1 \wedge \phi_2 \mid \neg \phi \mid \phi_1 \odot \phi_2 \mid [\phi]^{[\tau, \tau']}$$

where $\top$ stands for true, a is an atomic proposition in AP. The operators $\mathbf{H}^d$, $\odot$ and $[\ ]^{[\tau,\tau']}$ represent the hold operator with $d \in \mathbb{Z}_{\geq 0}$, concatenation operator and within operator respectively within a discrete-time constant interval $[\tau, \tau']$, where $\tau, \tau' \in \mathbb{Z}_{\geq 0}$ and $\tau' \geq \tau$, respectively and $\wedge$ and $\neg$ are the conjunction and negation operators respectively. The disjunction operator ($\vee$) can be derived from the negation and conjunction operators. Likewise, the implication operator ($\rightarrow$) can also be derived from the negation and disjunction operators.

TWTL semantics: The satisfaction relation defined by $\models$ defines when subtrace $t[i, j]$ of a (possibly infinite) timed-trace t from position i up to and including position j, satisfies the TWTL formula. This is denoted by $t[i, j] \models \phi$. The formula $\phi = \top$ always holds. The hold operator $\phi = \mathbf{H}^d a$ requires that a should hold for d time units. Likewise $\mathbf{H}^d \neg a$, specifies that for d time units, the condition $a \notin t[n].e$ should hold. The trace $t[i, j]$ satisfies the formulae $\phi = \phi_1 \wedge \phi_2$ when both subformulae are satisfied while in $\neg\phi$, $t[i, j]$, does not satisfy the given formula. A given formula in the form $\phi_1 \odot \phi_2$ specifies that the $t[i, j]$ should satisfy the first formula first and the second afterward with one time unit difference between the end of execution of ϕ_1 and start of execution of ϕ_2. The trace $t[i, j]$, must satisfy ϕ between the time window $[\tau, \tau']$ given $[\phi]^{[\tau,\tau']}$.

Given a TWTL formula ϕ and a timed-trace, $t[i, j]$ where the trace starts at $\tau_i \geq 0$ and terminates at $\tau_j \geq \tau_i$, the semantics of the operators is defined as:

$$
\begin{array}{lll}
t[i,j] \models \top & & \\
t[i,j] \models \mathbf{H}^d a & \text{iff} & a \in t[n].e, \forall n \in \{i,...,i+d\} \wedge \\
 & & (t[j].\tau - t[i].\tau) \geq d \\
t[i,j] \models \mathbf{H}^d \neg a & \text{iff} & a \notin t[n].e, \forall n \in \{i,...,i+d\} \wedge \\
 & & (t[j].\tau - t[i].\tau) \geq d \\
t[i,j] \models \phi_1 \wedge \phi_2 & \text{iff} & (t[i,j] \models \phi_1) \wedge (t[i,j] \models \phi_2) \\
t[i,j] \models \neg\phi & \text{iff} & \neg(t[i,j] \models \phi) \\
t[i,j] \models \phi_1 \odot \phi_2 & \text{iff} & \exists k = \arg\min_{i \leq k < j}\{t[i,k] \models \phi_1\} \wedge \\
 & & (t[k+1,j] \models \phi_2) \\
t[i,j] \models [\phi]^{[\tau,\tau']} & \text{iff} & \exists k \geq i+\tau,\ s.t.\ t[k,i+\tau'] \models \phi \wedge \\
 & & (t[j].\tau - t[i].\tau) \geq \tau'
\end{array}
$$

3 HYPERTWTL

A *hyperproperty* is a set of sets of infinite traces. HyperTWTL is a hyper-temporal logic to specify hyperproperties for TWTL [48] by extending the TWTL with quantification over multiple and concurrent execution traces. Hyperproperties can be specified with HyperTWTL using two different semantics, *synchronous* and *asynchronous*. Synchronous semantics requires timestamps in all quantified traces to match at each point in time. However, we can reason about traces that proceed at different speeds with asynchronous semantics. We first present the syntax and then the synchronous and asynchronous semantics of HyperTWTL.

3.1 Syntax of HyperTWTL

The syntax of HyperTWTL is inductively defined by the grammar as follows.

$$
\begin{aligned}
&\varphi := \exists\pi \cdot \varphi \mid \forall\pi \cdot \varphi \mid \phi \\
&\phi := \mathbf{H}^d a_\pi \mid \mathbf{H}^d \neg a_\pi \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \phi_1 \odot \phi_2 \mid [\phi]^{[\tau,\tau']} \mid \\
&\quad \mathbf{E}\rho \cdot \psi \mid \mathbf{A}\rho \cdot \psi \\
&\psi := \mathbf{H}^d a_{\pi,\rho} \mid \mathbf{H}^d \neg a_{\pi,\rho} \mid \psi_1 \wedge \psi_2 \mid \neg\psi \mid \psi_1 \odot \psi_2 \mid [\psi]^{S,T}
\end{aligned}
$$

where $a \in AP$, π is a trace variable from a set of trace variables $\mathcal{V}$ and ρ is a trajectory variable from the set $\mathcal{P}$. Thus, given $a_{\pi,\rho}$, the proposition $a \in AP$ holds in trace π and trajectory ρ at a given time point. The quantified formulae $\exists\pi$, and $\forall\pi$ are interpreted as "there exists some trace π" and "for all the traces π", respectively. The operators $\mathbf{H}^d$, $\odot$ and $[\ ]^{[\tau,\tau']}$ represent the hold operator with $d \in \mathbb{Z}_{\geq 0}$, concatenation operator and within operator respectively with a discrete-time constant interval $[\tau,\tau']$, where $\tau,\tau' \in \mathbb{Z}_{\geq 0}$ and $\tau' \geq \tau$, respectively and $\wedge$ and $\neg$ are the conjunction and negation operators respectively. Trace quantifiers $\exists\pi$ and $\forall\pi$, allow for the simultaneous reasoning about different traces. Similarly, trajectory quantifiers $\mathbf{E}\rho$ and $\mathbf{A}\rho$ allow reasoning simultaneously about different trajectories. The quantifier $\mathbf{E}$ is interpreted as there exists a trajectory that gives an interpretation of the relative passage of time between the traces for the inner temporal formula to be evaluated. Similarly, $\mathbf{A}$ means that all trajectories satisfy the inner formula. S and T are both intervals of form $[\tau,\tau']$ where $\tau,\tau' \in \mathbb{Z}_{\geq 0}$ and $\tau' \geq \tau$. The disjunction operator $(\vee)$ can be derived from the negation and conjunction operators. Likewise, the implication operator $(\rightarrow)$ can also be derived from the negation and disjunction operators.

3.2 Semantics of HyperTWTL

We classify the HyperTWTL formulae into two fragments based on the possible alternation in the HyperTWTL syntax as follows:

(1) Alternation-free HyperTWTL formulae with one type of quantifier, and
(2) k-alternation HyperTWTL formulae that allows k-alternation between existential and universal quantifiers.

3.2.1 *Synchronous Semantics of HyperTWTL.*

The satisfaction relation gives the semantics of synchronous HyperTWTL $\models_s$ over time-stamped traces $\mathbb{T}$. We define an assignment $\Pi : \mathcal{V} \rightarrow \mathcal{A}^\omega \times \mathbb{Z}_{\geq 0}$ as a partial function mapping trace variables to time-stamped traces. Let $\Pi(\pi) = (t,n)$ denote the time-stamped event from trace t at position n currently employed in the consideration of trace π. We then denote the explicit mapping of the trace variable π to a trace $t \in \mathbb{T}$ at position p as $\Pi[\pi \rightarrow (t,n)]$. Given (Π), where Π is the trace mapping, we use $(\Pi)+k$ as the k^{th} successor of (Π). We now present the synchronous semantics of HyperTWTL in Table 2.

In Table 2, the hold operator $\mathbf{H}^d a_\pi$ states that the proposition a is to be repeated for d time units in trace π. Similarly $\mathbf{H}^d \neg a_\pi$, requires that for d time units the proposition a should not be repeated in trace π. The trace set $\mathbb{T}$ satisfies both sub-formulae in $\phi = \phi_1 \wedge \phi_2$ while in $\neg\phi$, $\mathbb{T}$, does not satisfy the given formula. A given formula with a concatenation operator in the form $\phi_1 \odot \phi_2$ specifies that every $t \in \mathbb{T}$ should satisfy ϕ_1 first and then immediately ϕ_2 must also be satisfied with one-time unit difference between the end of execution of ϕ_1 and the start of execution of ϕ_2. The trace set, $\mathbb{T}$ must satisfy ϕ between the time window within the time window $[\tau,\tau']$ given $[\phi]^{[\tau,\tau']}$.

3.2.2 *Asynchronous Semantics of HyperTWTL.*

We now introduce the notion of a trajectory, which determines when traces move and when they stutter, to model the different speeds with which traces proceed in HyperTWTL. The asynchronous semantics of HyperTWTL is the obvious choice of semantics for applications with event-driven architectures. Let us denote $\mathcal{V}$ as a set of trace variables. Given a HyperTWTL formula φ, we denote $\mathbf{Paths}(\varphi)$ as a set of trace variables quantified in the formula φ. A given HyperTWTL formula φ is termed *asynchronous* if for all propositions $a_{\pi,\rho}$ in φ, π and ρ are quantified in φ. We require that for any given formula, no trajectory or trace variable is quantified twice. A *trajectory* $v : v_0v_1v_2\cdots$ for a given HyperTWTL formula is an infinite sequence of non-empty subsets of $\mathbf{Paths}(\varphi)$, i.e. $v \in \mathbf{Paths}(\varphi)$. Essentially, in each step of the trajectory, one or more of the traces may progress or all may stutter. A trajectory is fair for a trace variable $\pi \in \mathbf{Paths}$ if there are infinitely many positions j such that $\pi \in v_j$. Given a trajectory v, by v_i, we mean the suffix $v_iv_{i+1}v_{i+2}\cdots$. For a set of traces variables $\mathcal{V}$, we denote $\mathcal{R}_\mathcal{V}$ as the set of all fair trajectories for indices from $\mathcal{V}$. We use trace mapping Π as defined in the synchronous semantics of HyperTWTL. We now define the trajectory mapping $\Gamma : \mathbf{Vars}(\varphi) \rightarrow \mathcal{R}_{range(\Gamma)}$, where $range(\Gamma) \subset \mathbf{Vars}(\varphi)$ for which Γ is defined. We then denote the explicit mapping of the trajectory variable ρ to a trajectory v as $\Gamma[\rho \rightarrow v]$. Given (Π,Γ) where Π and Γ are the trace mapping and trajectory mapping respectively, we use $(\Pi,\Gamma)+k$ as the k^{th} successor of (Π,Γ). Given a trace mapping Π, a trace variable π, a

Table 2: Synchronous semantics of HyperTWTL

$(\mathbb{T}, \Pi) \models_s \exists\pi.\varphi$	iff	$\exists t \in \mathbb{T} \cdot (\mathbb{T}, \Pi[\pi \rightarrow (t, 0)]) \models_s \varphi$
$(\mathbb{T}, \Pi) \models_s \forall\pi.\varphi$	iff	$\forall t \in \mathbb{T} \cdot (\mathbb{T}, \Pi[\pi \rightarrow (t, 0)]) \models_s \varphi$
$(\mathbb{T}, \Pi) \models_s \mathbf{H}^d a_\pi$	iff	$a \in t[n].e$ for $(t, n) = \Pi(\pi), \forall n \in \{n, ..., n + d\} \wedge (t[n + i].\tau - t[n].\tau) \geq d$, for some $i > 0$
$(\mathbb{T}, \Pi) \models_s \mathbf{H}^d \neg a_\pi$	iff	$a \notin t[n].e$ for $(t, n) = \Pi(\pi), \forall n \in \{n, ..., n + d\} \wedge (t[n + i].\tau - t[n].\tau) \geq d$, for some $i > 0$
$(\mathbb{T}, \Pi) \models_s \phi_1 \wedge \phi_2$	iff	$((\mathbb{T}, \Pi) \models_s \phi_1) \wedge ((\mathbb{T}, \Pi) \models_s \phi_2)$
$(\mathbb{T}, \Pi) \models_s \neg\phi$	iff	$\neg((\mathbb{T}, \Pi) \models_s \phi)$
$(\mathbb{T}, \Pi) \models_s \phi_1 \odot \phi_2$	iff	$\exists k = \arg\min_{n \leq k \leq n'}$ for some $i \in [n, k] : ((\mathbb{T}, \Pi) \models_s \phi_1)$, for some $j \in [k + 1, n']$: $((\mathbb{T}, \Pi) \models_s \phi_2)$
$(\mathbb{T}, \Pi) \models_s [\phi]^{[\tau,\tau']}$	iff	$\exists k \geq n + \tau$, s.t. for all $i \in [k, n + \tau']$ $(\mathbb{T}, \Pi) \models_s \phi \ \wedge((\Pi) + n' - \Pi) \geq \tau'$ for some $n, n' \geq 0$

Table 3: Asynchronous semantics of HyperTWTL

$(\mathbb{T}, \Pi, \Gamma) \models_a \exists\pi.\varphi$	iff	$\exists t \in \mathbb{T} \cdot (\mathbb{T}, \Pi[\pi, \rho \rightarrow t, 0], \Gamma) \models_a \varphi$ for all ρ
$(\mathbb{T}, \Pi, \Gamma) \models_a \forall\pi.\varphi$	iff	$\forall t \in \mathbb{T} \cdot (\mathbb{T}, \Pi[\pi, \rho \rightarrow t, 0], \Gamma) \models_a \varphi$ for all ρ
$(\mathbb{T}, \Pi, \Gamma) \models_a \mathbf{E}\rho.\varphi$	iff	$\exists v \in \mathcal{R}_{range(\Gamma)} : (\mathbb{T}, \Pi, \Gamma[\rho \rightarrow v]) \models_a \varphi$
$(\mathbb{T}, \Pi, \Gamma) \models_a \mathbf{A}\rho.\varphi$	iff	$\forall v \in \mathcal{R}_{range(\Gamma)} : (\mathbb{T}, \Pi, \Gamma[\rho \rightarrow v]) \models_a \varphi$
$(\mathbb{T}, \Pi, \Gamma) \models_a \mathbf{H}^d a_{\pi,\rho}$	iff	$a \in t[n].e$ for $(t, n) = \Pi(\pi, \rho), \forall n \in \{n, ..., n + d\} \wedge (t[n + i].\tau - t[n].\tau) \geq d$, for some $i > 0$
$(\mathbb{T}, \Pi, \Gamma) \models_a \mathbf{H}^d \neg a_{\pi,\rho}$	iff	$a \notin t[n].e$ for $(t, n) = \Pi(\pi, \rho), \forall n \in \{n, ..., n + d\} \wedge (t[n + i].\tau - t[n].\tau) \geq d$, for some $i > 0$
$(\mathbb{T}, \Pi, \Gamma) \models_a \psi_1 \wedge \psi_2$	iff	$((\mathbb{T}, \Pi, \Gamma) \models_a \psi_1) \wedge ((\mathbb{T}, \Pi, \Gamma) \models_a \psi_2)$
$(\mathbb{T}, \Pi, \Gamma) \models_a \neg\psi$	iff	$\neg((\mathbb{T}, \Pi, \Gamma) \models_a \psi)$
$(\mathbb{T}, \Pi, \Gamma) \models_a \psi_1 \odot \psi_2$	iff	$\exists k = \arg\min_{n \leq k \leq n'}$, for some $i \in [n, k] : ((\mathbb{T}, \Pi, \Gamma) \models_a \psi_1)$, for some $j \in [k + 1, n']$: $((\mathbb{T}, \Pi, \Gamma) \models_a \psi_2)$
$(\mathbb{T}, \Pi, \Gamma) \models_a [\psi]^{S,T}$	iff	$\exists k \geq n + \tau$, s.t. for all $i \in [k, n + \tau']$: $(\mathbb{T}, \Pi, \Gamma) \models_a \psi \wedge [(\Pi, \Gamma) + n' - (\Pi, \Gamma)] \in S \wedge \lvert\Delta(\pi) - \Delta(\pi')\rvert \in T$, for some $n, n' > 0$

trajectory variable ρ, a trace t, and a pointer n, we denote the assignment that coincides with Π for every pair except for (π, ρ) which is mapped to (t, n) as $\Pi[(\pi, \rho) \rightarrow (t, n)]$. Given a HyperTWTL formula we denote Δ as the map from $\mathcal{V} \rightarrow \mathbb{Z}_{\geq 0}$ that returns the time duration for each trace variable $\pi \in range(\Delta)$. For all $\pi \in range(\Delta)$, we require that the following conditions be met:

- $(\Pi, \Gamma) + k - (\Pi, \Gamma) \in S$
- $|\Delta(\pi) - \Delta(\pi')| \in T$, for all distinct $\pi, \pi' \in range(\Delta)$

We, therefore, present the satisfaction of asynchronous semantics of HyperTWTL formula φ over trace mapping Π, trajectory mapping Γ, and a set of traces $\mathbb{T}$ denoted as $(\mathbb{T}, \Pi, \Gamma) \models_a \varphi$ in Table 3.

4 APPLICATIONS OF HYPERTWTL

In this section, we present a case study and illustrate some important requirements that can be expressed using HyperTWTL. This case study is inspired by the Technical Surveillance Squadron (TESS) of the United States Air Force Technical Applications Center [42], which provides persistent and collaborative surveillance of designated regions to detect, identify and locate potential nuclear explosions. The surveillance data is first gathered using autonomous security robots augmented with intelligent video cameras, onboard processors, communication modules, and navigation systems and then forwarded to the central control station for subsequent decision-making.

Fig. 1 shows the TKS of a TESS mission with different regions of interest and multiple robots. The surveillance environment is composed of 2 initial positions I_1 and I_2 (grey), 2 charging stations C_1 and C_2 (yellow), 6 regions of interest to be surveilled $R_1, \ldots, R_6$ (blue), and 10 allowable states $P_1, \ldots, P_{10}$ (white). The weighted transitions between the states represent the possible movements of agents in the environment, with respective weights representing the unit of time required for the transitions. The autonomous surveillance can be deployed with one or multiple robots starting from any initial states I_1 or I_2 with a fully charged battery within time bound $[0, T_1]$. The robot(s) then proceeds along the desired routes to perform surveillance at the regions of interest.

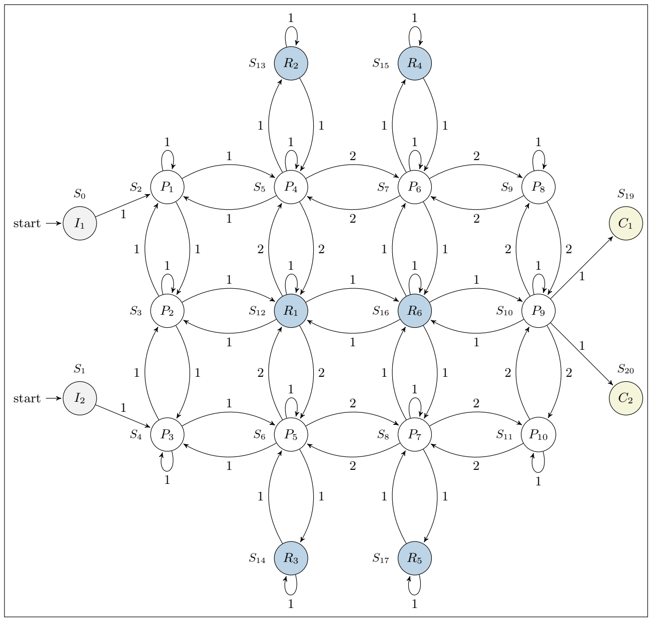

Figure 1: TKS of the TESS case study with initial states (grey), regions of interest (blue), charging states (yellow), and allowable states (white).

Each mission requires the surveillance of region R_1 within time bound $[T_2, T_3]$. Next, the robots are required to surveil either region R_2 followed by R_4 or region R_3 followed by R_5 within time bound $[T_4, T_5]$. Finally, a surveillance of region R_6 is performed within time bound $[T_6, T_7]$ before proceeding to a charging station C_1 or C_2 for recharging purposes within time bound $[T_8, T_9]$. Based on this case study, we consider several instances of hyperproperties that can be formalized as HyperTWTL formulae as follows.

Opacity: Information-flow security policies define what users can learn about a system while (partially) observing the system. Recent works show that robotic systems are prone to privacy/opacity attacks [17, 34]. A system is opaque if it meets two requirements: (i) there exists at least two executions of the system mapped to π_1 and π_2 with the same observations but bearing distinct secret, and (ii) the secret of each path cannot be accurately determined only by observing the system. Given a pair traces π_1 and π_2, let us assume the initial state I is the only information a system user can observe, and the surveillance routes are the secret to be kept from enemy forces. Then, we observe if the assigned task is performed on both traces while having different routes but the same observations O. This requirement can be formalized as a HyperTWTL formula φ_1 as shown in Table 4.

Non-interference: Non-interference is a security policy that seeks to restrict the flow of information within a system. This policy requires that low-security variables be independent of high-security variables, i.e., one should not be able to infer information about a high-security variable by observing low-security variables. Malicious software can disrupt the communication of robotic systems and access confidential messages exchanged on the system [35]. Given any pairs of executions from the case study above, let us assume that the initial state I is a high variable (high security) and paths from initial states to goal states denote a low variable (low security). The surveillance system satisfies non-interference if, there exists another execution π_2 that starts from a different high variable (i.e., the initial states are different), for all executions in π_1, and at the end of the mission, they are in the same low variable states (i.e., goal states). This requirement can be formalized as a HyperTWTL formula φ_2 as shown in Table 4.

Linearizability: The principle underlying linearizability is that the whole system operates as if executions from all robots are from one security robot. Thus, linearizability is a correctness condition to guarantee consistency across concurrent executions of a given system. Concurrency policies have been used to significantly enhance performance of robots applications [43, 44]. Any pair of traces must occupy the same states within the given mission time for the surveillance mission. At the same time, it is also important to ensure that the mission's primary goal to surveil either region R_2 followed by R_4 or R_3 followed by R_5 before proceeding to the charging state C_1 or C_2 is not violated. This can be formalized as a HyperTWTL formula φ_3 as shown in Table 4.

Mutation testing: Another interesting application of hyperproperty with quantifier alternation is the efficient generation of test cases for mutation testing. Let us assume that traces from all robots within the surveillance system are labeled as either *mutated* (t^m) or *non-mutated* ($t^{\neg m}$). We map t^m to π_1 and all other non-mutated traces $t^{\neg m}$ to π_2. This requirement guarantees that even if π_2 starts from the same initial state (I_1 or I_2) as π_1, they eventually proceed to different charging states (C_1 or C_2). This can be formalized as a HyperTWTL formula φ_4 as shown in Table 4.

Side-channel timing attacks: A side-channel timing attack is a security threat that attempts to acquire sensitive information from robotic applications by exploiting the execution time of the system. Recently, the robotic system's privacy, confidentiality, and availability have been compromised by side-channel timing attacks [3, 36]. To countermeasure this attack in our case study, it is required that each pair of mission execution (by a robot(s)), mapped to a pair of traces π_1 and π_2, and trajectories ρ and ρ', should end up in the charging state within close enough time after finishing their tasks. Let us assume that $[0, T_9]$ is the given time bound for the missions executions for π_1 and π_2, and the interval $[0, 2]$ specifies how close the mission execution times should be for π_1 and π_2. This requirement can be formalized as a HyperTWTL formula φ_5 as shown in Table 4.

Observational determinism: Observational determinism is another security policy that requires that for any given pair of traces π_1 and π_2 along the trajectory ρ, if the low-security inputs agree on both execution traces, then the low-security outputs must also agree in both traces. For example, given a set of executions from the case study presented above, let us assume that the initial state I is a low-security input and charging state C is a low-security output. The surveillance system then satisfies observation determinism if, for every pair of traces π_1 and π_2, the landing states, if the mission starts from the same initial state (I) for both traces, should be the same at the end of the mission within time bound $[0, T_9]$ with interval $[1, 3]$ specifying how close the execution time between π_1 and π_2 should be. This requirement can be formalized as a HyperTWTL formula φ_6 as shown in Table 4.

Service level agreements: A service level agreement (SLA) defines an acceptable performance of a given system. The expected specifications usually use statistics such as mean response time, time service factor, percentage uptime, etc. In this requirement, for any execution with a given response time π_1, we require that there has to exist another execution π_2 along a trajectory ρ with a similar timing behavior within the time bound $[T_8, T_9]$ with interval $[0, 2]$ specifying how close the execution time between π_1 and π_2 should be. This requirement can be formalized as a HyperTWTL formula φ_7 as shown in Table 4.

Note, all requirements in Table 4 are labeled as either Synchronous (Sync) or Asynchronous (Async).

5 MODEL CHECKING OF HYPERTWTL

Given a TKS $\mathcal{T}$ and a HyperTWTL formula φ, the model checking problem checks whether $\mathcal{T} \models_s \varphi$. In the later sections, we discuss the decidability of the HyperTWTL model checking problem for both the fragments, alternation-free HyperTWTL and k-alternation

Algorithm 1: Model checking of alternation-free HyperTWTL

```
Inputs  : HyperTWTL formula $\varphi$, TKS $\mathcal{T}$
Outputs : Verdict = $\bot, \top$
1: if (is_synchronous($\varphi$)) then
2:     $\hat{\varphi} \leftarrow \varphi$
3: else
4:     $\hat{\varphi} \leftarrow$ Asynch_to_Synch($\varphi$)
5: end if
6: $\varphi_{TWTL} \leftarrow$ HyperTWTL_to_PlainTWTL ($\hat{\varphi}$)
7: $\mathcal{T}' \leftarrow$ ModelGen($\mathcal{T}, \varphi$)
8: $\beta \leftarrow$ Verify($\mathcal{T}', \varphi_{TWTL}$)
9: return $\beta$
```

HyperTWTL, and demonstrate the method of translating an asynchronous HyperTWTL formula to a synchronous formula and translating a HyperTWTL formula into an equivalent TWTL formula. We restrict the model checking of HyperTWTL to the decidable alternation-free fragments ($\forall^*$- or $\exists^*$-) and 1-alternation fragments ($\exists^*\forall^*$). However, we do not allow fragments of HyperTWTL where a nesting structure of temporal logic formulae involves different traces.

Self-composition. Let $\varphi = Q.\phi$ be a HyperTWTL formula which describes a TKS $\mathcal{T}$, where Q is a block of quantifiers and ϕ is the inner TWTL formula. To assert that $\mathcal{T} \models \varphi$, we generate the system $\mathcal{T}'$ which has n copies of the system $\mathcal{T}$ running in parallel consisting of traces over $\mathcal{A}$. Thus, given a 2-fold parallel self-composition of $\mathcal{T}$, we define $\mathcal{T}'$ as: $\mathcal{T}' = \mathcal{T}_1 \times \mathcal{T}_2 \triangleq \{(t_0, t'_0), (t_1, t'_1) \cdots \mid t \in \mathbb{T} \wedge t' \in \mathbb{T}\}$

Proposition 1. Given a HyperTWTL formula φ, if there exists an equivalent TWTL formula φ_{TWTL}, then $\mathcal{T} \models_s \varphi \Leftrightarrow \mathcal{T}' \models_s \varphi_{TWTL}$.

Proof. Let $\mathbb{T}$ be a set of traces generated over the model $\mathcal{T}$, and $\mathbb{T}'$ be a set of traces generated over the model $\mathcal{T}'$, so $\mathcal{T}'$ contains n copies of $\mathcal{T}$. Thus, for any set of traces $\Pi \subseteq \mathbb{T}$ that satisfies the HyperTWTL formula φ, there exists a set of traces $\Pi' \subseteq \mathbb{T}'$ such that Π' satisfies the equivalent TWTL formula φ_{TWTL}, where all traces in Π are in Π', with unique fresh names from $\mathcal{T}'$.

5.1 Model checking alteration-free HyperTWTL

We present Algorithm 1 illustrating the overall model checking approach given a TKS $\mathcal{T}$ and an alternation-free HyperTWTL formula. The steps are described as follows.

a) First, our model checking algorithm checks if the input HyperTWTL formula φ is synchronous or not. (Line 1)
b) If φ is an asynchronous formula, we translate the asynchronous HyperTWTL formula to an equivalent synchronous HyperTWTL formula using the function `Asynch_to_Synch()` in (Line 4), explained in detail in the next paragraph.
c) Then, we transform the synchronous HyperTWTL formula $\hat{\varphi}$ into an equivalent TWTL formula φ_{TWTL} using the function `HyperTWTL_to_PlainTWTL()` (Line 6).
d) Using a function `ModelGen()`, we generate a new model that contains copies of the original model through the process of self-composition [7]. The number of copies equals the number of quantifiers of the formula φ or ψ (Line 7).
e) Next, the `Verify()` function takes as inputs the new model $\mathcal{T}$' and the equivalent TWTL formula φ_{TWTL}, and then utilizes the verification approach from [48] to solve the model checking problem (Line 8).
f) Finally, we return the verdict in case of satisfaction/violation ($\top/\bot$) (Line 9).

5.2 Asynchronous HyperTWTL to Synchronous HyperTWTL

The process to convert a given asynchronous HyperTWTL formula to a synchronous HyperTWTL formula has two parts. First, we generate *invariant* set of traces $inv(\mathbb{T})$ for the corresponding trace set $\mathbb{T}$ generated over model $\mathcal{T}$. This allows for the synchronization of interleaving traces while reconciling the synchronous and asynchronous semantics of HyperTWTL. Secondly, we construct an equivalent synchronous formula $\hat{\varphi}$ from an asynchronous formula φ such that $\mathbb{T} \models_a \varphi$ if and only if $inv(\mathbb{T}) \models_s \hat{\varphi}$. These steps are described as follows.

5.2.1 *Invariant Trace Generation.*

To construct an equivalent HyperTWTL synchronous formula $\hat{\varphi}$ from a given asynchronous HyperTWTL formula φ, we require that HyperTWTL be *stutter insensitive*[39]. To achieve this, we define the variable γ_π^ρ needed for the evaluation of the atomic propositions across traces. Thus, given a pair of traces π_1 and π_2, γ_π^ρ ensures that all propositions in both traces exhibit the identical sequence at all timestamps. However, since timestamps proceed at different speeds in different traces such as π_1 and π_2, a trajectory ρ is used to determine which trace moves and which trace stutters at any time point. In an attempt to synchronize traces once non-aligned timestamps are identified by a trajectory, silent events (ϵ) are introduced between the time stamps of the trace. For all $t \in \mathbb{T}$, we denote $inv(\mathbb{T})$ as the maximal set of traces defined over $\mathcal{A}_\epsilon$ where $\mathcal{A}_\epsilon = \mathcal{A} \cup \epsilon$. Consider a trace $t = (3, \{b\})(6, \{a\})(8, \{b\}) \cdots$. The trace $t' \in inv(\mathbb{T})$ can be generated as $inv(t) = \epsilon\epsilon\epsilon b\epsilon\epsilon a\epsilon b \cdots$. We now construct the synchronous HyperTWTL formula to reason about the trace set $inv(\mathbb{T})$.

5.2.2 *Synchronous HyperTWTL Formula Construction.*

We now construct a synchronous formula $\hat{\varphi}$ that is equivalent to the asynchronous HyperTWTL φ. Intuitively, the asynchronous formula of HyperTWTL φ depends on a finite interval of a timed trace. Thus, we can replace the asynchronous formula φ with a synchronous formula $\hat{\varphi}$ that encapsulates the interval patterns in the asynchronous formula φ. Given a bounded asynchronous formula φ, we define β_φ as the projected time period required to satisfy the asynchronous formula. Inductively, β_φ can be defined as: $\beta_{\mathbf{H}^d\ a} = d$ for the $\mathbf{H}$ operator; $\beta_{\varphi_1 \wedge \varphi_2} = max(\beta_{\varphi_1}, \beta_{\varphi_2})$ for the $\wedge$ operator; $\beta_{\neg\varphi} = \beta_\varphi$ for the $\neg$ operator; $\beta_{\varphi_1 \odot \varphi_2} = \beta_{\varphi_1} + \beta_{\varphi_2} + 1$ for the $\odot$ operator; $\beta_{[\varphi]^{S,T}} = up(S) + up(T)$ for the [ ] operator, where $up \rightarrow \mathbb{Z}_{\geq 0}$ returns the upper bound of a predefined time bound. We then construct a synchronous formula $\hat{\varphi}$ from an asynchronous formula φ by replacing the time required for the satisfaction of φ with the appropriate ρ_φ.

Proposition 2. Given a set of traces $\mathbb{T}$ and an alternation-free asynchronous HyperTWTL formula φ over $\mathcal{A}$, $\mathbb{T} \models \varphi$ iff $inv(\mathbb{T}) \models_s \hat{\varphi}$.

Given $\mathbb{T}$ is a set of traces generated over TKS, let $inv(\mathbb{T})$ denote the extended set of traces for $\mathbb{T}$ where ϵ has been introduced between occurrences of events to synchronize events of each $t' \in inv(\mathbb{T})$. The trace t can be constructed from t' by deleting the ϵ-events in t'. Given the synchronous HyperTWTL formula $\hat{\varphi}$ does not violate the interval patterns of the associated asynchronous formula ψ, we can conclude that, if $\mathbb{T} \models \varphi$, then $inv(\mathbb{T}) \models_s \hat{\varphi}$.

5.3 Converting HyperTWTL to TWTL formula

We verify the HyperTWTL specifications by creating a new model that contains copies of the original system, where the number of copies is equal to the number of quantifiers in the HyperTWTL formula. The new model is then verified against a given HyperTWTL specification. Given a HyperTWTL formula φ, let $\phi_1 \dots \phi_{N \in \mathbb{Z}_{\geq 0}}$ be the sub-formulae of φ, where the same proposition is observed on any pair of traces. We denote $M^j_{\phi_i}$ as an instance of ϕ_i where $i \leq N$ and $j \in \mathbb{Z}_{\geq 0}$ is the index of the copy of the original model. For example, consider the HyperTWTL formula φ_1 in Table 4. The following sub-formulae of φ_1 analyze the same proposition on the given pair of traces π_1 and π_2: $\phi_1 = [\mathbf{H}^1\ I_{\pi_1}\ \wedge\ \mathbf{H}^1\ I_{\pi_2}]^{[0,T_1]}$, $\phi_2 = [\mathbf{H}^1\ R_{1\pi_1}\ \wedge\ \mathbf{H}^1\ R_{1\pi_2}]^{[T_2,T_3]}$, $\phi_3 = [\mathbf{H}^1\ R_{6\pi_1}\ \wedge\ \mathbf{H}^1\ R_{6\pi_2}]^{[T_6,T_7]}$, and $\phi_4 = [\mathbf{H}^{T_7-T_2}\ O_{\pi_1} = \mathbf{H}^{T_7-T_2}\ O_{\pi_2}]^{[T_2,T_7]}$ We obtain an equivalent TWTL formula by removing the quantifier prefix and introducing fresh atomic propositions that capture the notion of occurrences of the observation of the same proposition on any pair of traces. In the case where different propositions are to be observed on a given pair of traces, we maintain the proposition while introducing a superscript $j \in \mathbb{Z}_{\geq 0}$ for the same purpose as described above. Given φ_1 has two quantifiers, two copies of the original model will be needed to verify the equivalent TWTL formula. Thus, we denote $M^1_{\phi_1}$ and $M^2_{\phi_1}$ as instances of ϕ_1 for the first and second copies of the model. Similarly, $M^1_{\phi_3}$ and $M^2_{\phi_3}$ are the first and second copies of the model for the instance of ϕ_3. The HyperTWTL formula φ_1 can then be translated as an equivalent TWTL as $\theta_1 = ([\mathbf{H}^1 M^1_{\phi_1}]^{[0,T_1]} \odot [\mathbf{H}^1 M^1_{\phi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1 R^1_2 \odot \mathbf{H}^1 R^1_4]^{[T_4,T_5]} \vee [\mathbf{H}^1 R^1_3\ \odot \mathbf{H}^1 R^1_5]^{[T_4,T_5]}) \odot [\mathbf{H}^1 M^1_{\phi_3}]^{[T_6,T_7]} \wedge [\mathbf{H}^{T_7-T_2} M^1_{\phi_4}]^{[T_7,T_2]}) \wedge ([\mathbf{H}^1 M^2_{\phi_1}]^{[0,T_1]} \odot [\mathbf{H}^1 M^2_{\phi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1 R^2_2 \odot \mathbf{H}^1 R^2_4]^{[T_4,T_5]} \vee [\mathbf{H}^1 R^2_3 \odot \mathbf{H}^1 R^2_5]^{[T_4,T_5]}) \odot [\mathbf{H}^1 M^2_{\phi_3}]^{[T_6,T_7]} \wedge [\mathbf{H}^{T_7-T_2} M^2_{\phi_4}]^{[T_7,T_2]})$.

In the case of φ_5 (since it is an asynchronous formula), we first translate the formula into an equivalent synchronous HyperTWTL formula using the method we described in Section 5(B). The resulting synchronous formula is then translated into an equivalent TWTL formula using the same method described in the previous example of φ_1 and is shown in Table 5. Compared to φ_1 and φ_5, the translation of φ_2, φ_3, φ_4, φ_6, and φ_7 to equivalent TWTL formulae is not straightforward. This is because the model checking problem of HyperTWTL approaches undecidability when only a single quantifier alternation is allowed [30]. However, model checking of HyperTWTL formulae of the form $\exists^*\forall^*$ is decidable when the universal quantifier is flattened. For instance, consider the HyperTWTL formula φ_3 in Table 4. To solve this formula, we flatten the literals associated with π_2 by enumerating all the possible interactions between π_1 and π_2, thus reducing the problem to a $\exists^*\exists^*$ problem. Let us denote $\psi_1 \dots \psi_{N \in \mathbb{Z}_{\geq 0}}$ as sub-formulae of φ_3 which encapsulates all the possible interactions between π_1 and π_2. Thus, the first sub-formula of φ_3, $[H^1 I_{\pi_1} = H^1 I_{\pi_2}]^{[0,T_1]}$ yields $\psi_1 = [H^1 I_{1\pi_1} \wedge H^1 I_{2\pi_2}]^{[0,T_1]} \vee [H^1 I_{2\pi_1} \wedge H^1 I_{1\pi_2}]^{[0,T_1]}$. We then denote $B^j_{\psi_i}$ as an instance of ψ_i where $i \leq N$ and $j \in \mathbb{Z}_{\geq 0}$ is the index of the copy of the original model. Once the alternation is eliminated, φ_3 and φ_4 can be converted to equivalent TWTL formulae using the same approach described for φ_1 and φ_2 as shown in Table 5.

5.4 Model checking k-alternation HyperTWTL

Model checking k-alternations formulae are generally complex as all given executions have to be examined. For instance, consider the HyperTWTL formula $\varphi = \exists\pi_1.\forall\pi_2 \cdot \phi$. To verify this formula, it requires that for all traces $t \in \mathbb{T}$, there exists a trace t that the formula ϕ is violated. The situation is dire in specifications with more than one alternation of quantifiers. Model checking such specifications may lead to potential state explosion, even with a finite set of traces. Consistent with the general notion of undecidability of a model checking problem, $\exists^*\forall^*$ and $\forall^*\exists^*$ fragments of HyperTWTL are undecidable in both the synchronous and asynchronous semantics. Despite the difficulty and complexity of model checking of HyperTWTL beyond the alternation-free fragments, k-alternation fragment of HyperTWTL can be decided within a bounded time domain as follows.

The model checking for HyperTWTL becomes decidable when there is an *a priori* bound k_{lim} on the variability of the traces. The variability of a timed trace is the maximum possible number of events in any open unit interval. Hence, given the bound $k_{lim} \in \mathbb{Z}_{\geq 0}$, the number of events in a timed trace is less than or equal to k_{lim}. Therefore, the verification of synchronous HyperTWTL (at least for the $\exists^*\forall^*$-fragment) can be decided with any tool that works with TWTL. With asynchronous HyperTWTL ψ, given a set of traces $\mathbb{T}$ and a variability bound k_{lim}, we only evaluate traces whose timestamps are less than k_{lim}. Let $\mathbb{T}_{[0,k_{lim})}$ denote the set of all such traces in $\mathbb{T}$, i.e. $\mathbb{T}_{[0,k_{lim})} \subseteq \mathbb{T}$. We can now reduce the model checking problem to $\mathbb{T}_{[0,k_{lim})} \models_s \psi$.

Proposition 3. Model checking HyperTWTL can be decided when all traces have constrained variability k_{lim} where $k_{lim} \in \mathbb{Z}_{\geq 0}$.

5.5 Complexity

In this section, we review the complexity of Algorithm 1 for the model checking of alternation-free HyperTWTL formulae. The time complexity of translating an asynchronous HyperTWTL to synchronous HyperTWTL formula is based on the structure of the formula. Translating the asynchronous formula to a synchronous formula in Algorithm 1 takes $O(|\varphi|)$ at most. The time complexity of translating HyperTWTL to TWTL is upper-bounded to $O(|\varphi| \cdot 2^{|AP|})$. The model generation function in Algorithm 1 depends on the structure of the formula and the number of quantifiers in the given formula. The time complexity of the model generation is $O(|Q|^n)$, where n is the number of copies of the original model. The satisfiability problem of HyperTWTL can be solved similarly as for HyperLTL, i.e., satisfiability is decidable for HyperTWTL fragments not containing $\forall\exists$ irrespective of the semantics used. Hence, the

Table 4: Requirements expressed in HyperTWTL

No.	Description	Type	HyperTWTL Specification
1	Opacity	Synch.	$\varphi_1 = \exists\pi_1\exists\pi_2 \cdot [\mathbf{H}^1\, I_{\pi_1} \wedge \mathbf{H}^1\, I_{\pi_2}]^{[0,T_1]} \odot ([\mathbf{H}^1\, R_{1\pi_1} \wedge \mathbf{H}^1\, R_{1\pi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1\, R_{2\pi_1} \odot \mathbf{H}^1\, R_{4\pi_1}]^{[T_4,T_5]} \wedge [\mathbf{H}^1\, R_{3\pi_2} \odot \mathbf{H}^1\, R_{5\pi_2}]^{[T_4,T_5]}) \odot [\mathbf{H}^1\, R_{6\pi_1} \wedge \mathbf{H}^1\, R_{6\pi_2}]^{[T_6,T_7]}) \wedge [\mathbf{H}^{T_7-T_2}\, O_{\pi_1} = \mathbf{H}^{T_7-T_2}\, O_{\pi_2}]^{[T_2,T_7]}$
2	Non-Interference	Synch.	$\varphi_2 = \exists\pi_1\forall\pi_2 \cdot [\mathbf{H}^1\, I_{\pi_1} \neq \mathbf{H}^1\, I_{\pi_2}]^{[0,T_1]} \rightarrow ([\mathbf{H}^1\, R_{1\pi_1} \wedge \mathbf{H}^1\, R_{1\pi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1\, R_{2\pi_1} \odot \mathbf{H}^1\, R_{4\pi_1}]^{[T_4,T_5]} \wedge [\mathbf{H}^1\, R_{3\pi_2} \odot \mathbf{H}^1\, R_{5\pi_2}]^{[T_4,T_5]}) \odot [\mathbf{H}^1\, R_{6\pi_1} \wedge \mathbf{H}^1\, R_{6\pi_2}]^{[T_6,T_7]}) \odot [\mathbf{H}^1\, C_{\pi_1} = \mathbf{H}^1\, C_{\pi_2}]^{[T_8,T_9]}$
3	Lineariz-ability	Synch.	$\varphi_3 = \exists\pi_1\forall\pi_2 \cdot [\mathbf{H}^1\, I_{\pi_1} = \mathbf{H}^1\, I_{\pi_2}]^{[0,T_1]} \odot ([\mathbf{H}^1\, R_{1\pi_1} \wedge \mathbf{H}^1\, R_{1\pi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1\, R_{2\pi_1} \odot \mathbf{H}^1\, R_{4\pi_1}]^{[T_4,T_5]} \wedge [\mathbf{H}^1\, R_{3\pi_2} \odot \mathbf{H}^1\, R_{5\pi_2}]^{[T_4,T_5]}) \odot [\mathbf{H}^1\, R_{6\pi_1} \wedge \mathbf{H}^1\, R_{6\pi_2}]^{[T_6,T_7]} \wedge [\mathbf{H}^{T_7-T_2}\, P_{\pi_1} = \mathbf{H}^{T_7-T_2}\, P_{\pi_2}]^{[T_2,T_7]}) \odot [\mathbf{H}^1\, C_{\pi_1} = \mathbf{H}^1\, C_{\pi_2}]^{[T_8,T_9]}$
4	Mutation Testing	Synch.	$\varphi_4 = \exists\pi_1\forall\pi_2 \cdot [\mathbf{H}^d\, t^m_{\pi_1} \wedge \mathbf{H}^d\, t^{\neg m}_{\pi_2}]^{[0,T_9]} \wedge [\mathbf{H}^1\, I_{\pi_1} = \mathbf{H}^1\, I_{\pi_2}]^{[0,T_1]} \odot ([\mathbf{H}^1\, R_{1\pi_1} \wedge \mathbf{H}^1\, R_{1\pi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1\, R_{2\pi_1} \odot \mathbf{H}^1\, R_{4\pi_1}]^{[T_4,T_5]} \wedge [\mathbf{H}^1\, R_{3\pi_2} \odot \mathbf{H}^1\, R_{5\pi_2}]^{[T_4,T_5]}) \odot [\mathbf{H}^1\, R_{6\pi_1} \wedge \mathbf{H}^1\, R_{6\pi_2}]^{[T_6,T_7]}) \odot [\mathbf{H}^1\, C_{\pi_1} \neq \mathbf{H}^1\, C_{\pi_2}]^{[T_8,T_9]}$, where $d = T_9$
5	Side-Channel Timing Attacks	Asynch.	$\varphi_5 = \forall\pi_1\forall\pi_2 \cdot \mathbf{A}\rho\mathbf{E}\rho' \cdot [\mathbf{H}^1\, I_{\pi_1,\rho} \wedge \mathbf{H}^1\, I_{\pi_2,\rho'}]^{[0,T_1]} \rightarrow ([\mathbf{H}^1\, R_{1\pi_1,\rho} \wedge \mathbf{H}^1\, R_{1\pi_2,\rho'}]^{[T_2,T_3]} \odot ([\mathbf{H}^1\, R_{2\pi_1,\rho} \odot \mathbf{H}^1\, R_{4\pi_1,\rho}]^{[T_4,T_5]} \wedge [\mathbf{H}^1\, R_{3\pi_2,\rho'} \odot \mathbf{H}^1\, R_{5\pi_2,\rho'}]^{[T_4,T_5]}) \odot [\mathbf{H}^1\, R_{6\pi_1,\rho} \wedge \mathbf{H}^1\, R_{6\pi_2,\rho'}]^{[T_6,T_7]}) \odot [\mathbf{H}^1\, C_{\pi_1,\rho} \wedge \mathbf{H}^1\, C_{\pi_2,\rho'}]^{[T_8,T_9],[0,2]}$
6	Observational Determinism	Asynch.	$\varphi_6 = \exists\pi_2\forall\pi_1 \cdot \mathbf{A}\rho \cdot [\mathbf{H}^1\, I_{\pi_1,\rho} = \mathbf{H}^1\, I_{\pi_2,\rho}]^{[0,T_1]} \rightarrow ([\mathbf{H}^1\, R_{1\pi_1,\rho} \wedge \mathbf{H}^1\, R_{1\pi_2,\rho}]^{[T_2,T_3]} \odot ([\mathbf{H}^1\, R_{2\pi_1,\rho} \odot \mathbf{H}^1\, R_{4\pi_1,\rho}]^{[T_4,T_5]} \wedge [\mathbf{H}^1\, R_{3\pi_2,\rho} \odot \mathbf{H}^1\, R_{5\pi_2,\rho}]^{[T_4,T_5]}) \odot [\mathbf{H}^1\, R_{6\pi_1,\rho} \wedge \mathbf{H}^1\, R_{6\pi_2,\rho}]^{[T_6,T_7]}) \odot [\mathbf{H}^1\, C_{\pi_1,\rho} = \mathbf{H}^1\, C_{\pi_2,\rho}]^{[T_8,T_9][1,3]}$
7	Service Level Agreement	Asynch.	$\varphi_7 = \exists\pi_2\forall\pi_1 \cdot \mathbf{E}\rho \cdot [\mathbf{H}^1\, I_{1\pi_1,\rho} \wedge \mathbf{H}^1\, I_{1\pi_2,\rho}]^{[0,T_1]} \rightarrow [\mathbf{H}^1\, C_{\pi_1,\rho} \wedge \mathbf{H}^1\, C_{\pi_2,\rho}]^{[T_8,T_9],[0,2]}$

Table 5: Equivalent TWTL formulae of HyperTWTL in Table 4

No.	TWTL Specifications
1	$\theta_1 = ([\mathbf{H}^1 M^1_{\phi_1}]^{[0,T_1]} \odot [\mathbf{H}^1 M^1_{\phi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1 R^1_2 \odot \mathbf{H}^1 R^1_4]^{[T_4,T_5]} \vee [\mathbf{H}^1 R^1_3 \odot \mathbf{H}^1 R^1_5]^{[T_4,T_5]}) \odot [\mathbf{H}^1 M^1_{\phi_3}]^{[T_6,T_7]} \wedge [\mathbf{H}^{T_2-T_7} M^1_{\phi_4}]^{[T_7,T_2]}) \wedge ([\mathbf{H}^1 M^2_{\phi_1}]^{[0,T_1]} \odot [\mathbf{H}^1 M^2_{\phi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1 R^2_2 \odot \mathbf{H}^1 R^2_4]^{[T_4,T_5]} \vee [\mathbf{H}^1 R^2_3 \odot \mathbf{H}^1 R^2_5]^{[T_4,T_5]}) \odot [\mathbf{H}^1 M^2_{\phi_3}]^{[T_6,T_7]} \wedge [\mathbf{H}^{T_7-T_2} M^2_{\phi_4}]^{[T_2,T_7]})$
2	$\theta_2 = ([\mathbf{H}^1 I^1_{\psi_1}]^{[0,T_1]} \rightarrow [\mathbf{H}^1 B^1_{\psi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1 R^1_2 \odot \mathbf{H}^1 R^1_4]^{[T_4,T_5]} \vee [\mathbf{H}^1 R^1_3 \odot \mathbf{H}^1 R^1_5]^{[T_4,T_5]}) \odot [\mathbf{H}^1 B^1_{\psi_3}]^{[T_6,T_7]} \odot [\mathbf{H}^1 B^1_{\psi_4}]^{[T_8,T_9]}) \wedge ([\mathbf{H}^1 B^2_{\psi_1}]^{[0,T_1]} \rightarrow [\mathbf{H}^1 B^2_{\psi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1 R^2_2 \odot \mathbf{H}^1 R^2_4]^{[T_4,T_5]} \vee [\mathbf{H}^1 R^2_3 \odot \mathbf{H}^1 R^2_5]^{[T_4,T_5]}) \odot [\mathbf{H}^1 B^2_{\psi_3}]^{[T_6,T_7]} \odot [\mathbf{H}^1 B^2_{\psi_4}]^{[T_8,T_9]})$
3	$\theta_3 = ([\mathbf{H}^1 B^1_{\psi_1}]^{[0,T_1]} \odot [\mathbf{H}^1 B^1_{\psi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1 R^1_2 \odot \mathbf{H}^1 R^1_4]^{[T_4,T_5]} \vee [\mathbf{H}^1 R^1_3 \odot \mathbf{H}^1 R^1_5]^{[T_4,T_5]}) \odot [\mathbf{H}^1 B^1_{\psi_3}]^{[T_6,T_7]} \odot [\mathbf{H}^1 B^1_{\psi_4}]^{[T_8,T_9]} \wedge [\mathbf{H}^{T_7-T_2} B^1_{\psi_5}]^{[T_2,T_7]}) \vee ([\mathbf{H}^1 B^2_{\psi_1}]^{[0,T_1]} \odot [\mathbf{H}^1 B^2_{\psi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1 R^2_3 \odot \mathbf{H}^1 R^2_5]^{[T_4,T_5]} \vee [\mathbf{H}^1 R^2_3 \odot \mathbf{H}^1 R^2_5]^{[T_4,T_5]}) \odot [\mathbf{H}^1 B^2_{\psi_3}]^{[T_6,T_7]} \odot [\mathbf{H}^1 B^2_{\psi_4}]^{[T_8,T_9]} \wedge [\mathbf{H}^{T_7-T_2} B^2_{\psi_5}]^{[T_2,T_7]})$
4	$\theta_3 = ([\mathbf{H}^1 B^1_{\psi_1}]^{[0,T_1]} \wedge [\mathbf{H}^1 B^1_{\psi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1 R^1_2 \odot \mathbf{H}^1 R^1_4]^{[T_4,T_5]} \vee [\mathbf{H}^1 R^1_3 \odot \mathbf{H}^1 R^1_5]^{[T_4,T_5]}) \odot [\mathbf{H}^1 B^1_{\psi_3}]^{[T_6,T_7]} \odot [\mathbf{H}^1 B^1_{\psi_4}]^{[T_8,T_9]}) \vee ([\mathbf{H}^1 B^2_{\psi_1}]^{[0,T_1]} \odot [\mathbf{H}^1 B^2_{\psi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1 R^2_2 \odot \mathbf{H}^1 R^2_4]^{[T_4,T_5]} \vee [\mathbf{H}^1 R^2_3 \odot \mathbf{H}^1 R^2_5]^{[T_4,T_5]}) \odot [\mathbf{H}^1 B^2_{\psi_3}]^{[T_6,T_7]} \odot [\mathbf{H}^1 B^2_{\psi_4}]^{[T_8,T_9]})$
5	$\theta_5 = ([\mathbf{H}^1 M^1_{\phi_1}]^{[0,T_1]} \rightarrow [\mathbf{H}^1 M^1_{\phi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1 R^1_2 \odot \mathbf{H}^1 R^1_4]^{[T_4,T_5]} \vee [\mathbf{H}^1 R^1_3 \odot \mathbf{H}^1 R^1_5]^{[T_4,T_5]}) \odot [\mathbf{H}^1 M^1_{\phi_3}]^{[T_6,T_7]} \odot [\mathbf{H}^1 M^1_{\phi_4}]^{[T_8,T_9]}) \wedge ([\mathbf{H}^1 M^2_{\phi_1}]^{[0,T_1]} \rightarrow [\mathbf{H}^1 M^2_{\phi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1 R^2_2 \odot \mathbf{H}^1 R^2_4]^{[T_4,T_5]} \vee [\mathbf{H}^1 R^2_3 \odot \mathbf{H}^1 R^2_5]^{[T_4,T_5]}) \odot [\mathbf{H}^1 M^2_{\phi_3}]^{[T_6,T_7]} \odot [\mathbf{H}^1 M^2_{\phi_4}]^{[T_8,T_9]})$
6	$\theta_6 = ([\mathbf{H}^1 B^1_{\psi_1}]^{[0,T_1]} \rightarrow [\mathbf{H}^1 B^1_{\psi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1 R^1_2 \odot \mathbf{H}^1 R^1_4]^{[T_4,T_5]} \vee [\mathbf{H}^1 R^1_3 \odot \mathbf{H}^1 R^1_5]^{[T_4,T_5]}) \odot [\mathbf{H}^1 B^1_{\psi_3}]^{[T_6,T_7]} \odot [\mathbf{H}^1 B^1_{\psi_4}]^{[T_8,T_9]}) \wedge ([\mathbf{H}^1 B^2_{\psi_1}]^{[0,T_1]} \rightarrow [\mathbf{H}^1 B^2_{\psi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1 R^2_2 \odot \mathbf{H}^1 R^2_4]^{[T_4,T_5]} \vee [\mathbf{H}^1 R^2_3 \odot \mathbf{H}^1 R^2_5]^{[T_4,T_5]}) \odot [\mathbf{H}^1 B^2_{\psi_3}]^{[T_6,T_7]} \odot [\mathbf{H}^1 B^2_{\psi_4}]^{[T_8,T_9]})$
7	$\theta_7 = ([\mathbf{H}^1 M^1_{\phi_1}]^{[0,T_1]} \rightarrow [\mathbf{H}^1 M^1_{\phi_2}]^{[T_8,T_9]}) \wedge ([\mathbf{H}^1 M^2_{\phi_1}]^{[0,T_1]} \rightarrow [\mathbf{H}^1 M^2_{\phi_2}]^{[T_8,T_9]})$

complexity results for the various fragments of HyperTWTL will then be similar to that of HyperLTL, i.e., the satisfiability problem for the alternation-free fragment and bounded $\exists^*\forall^*$ fragments of HyperTWTL is thus PSPACE-complete.

6 IMPLEMENTATION AND RESULTS

To illustrate the effectiveness of our proposed methods, in this section, we present the evaluation of the TESS case study described in Section 4 using Algorithm 1. The HyperTWTL specifications

Table 6: Experimental results for model checking TWTL formulae in Table 5

TWTL Specification	Verdict	Time (Seconds)	Memory (MB)
θ_1	SAT	16.30	15.21
θ_2	UNSAT	19.05	15.33
θ_3	UNSAT	22.24	17.25
θ_4	UNSAT	24.52	16.27
θ_5	SAT	25.19	16.04
θ_6	UNSAT	23.83	16.39
θ_7	SAT	22.24	17.48

for this case study are presented in Table 4 where $\varphi_1 - \varphi_4$ are synchronous formulae and $\varphi_5 - \varphi_7$ are asynchronous formulae. All formulae were converted into equivalent TWTL formulae (see Section 5) as shown in Table 5. For verifying the formulae in Table 5, we use the PyTWTL tool [47], a Python 2.7 implementation of the TWTL verification algorithms proposed in [48]. All the experiments are performed on a Windows 10 system with 64 GB RAM and Intel Core(TM) i9-10900 CPU (3.70 GHz). The following time bounds are considered for the verification of all the equivalent flat TWTL properties in Table 5: $T_1 = 2,\ T_2 = 5,\ T_3 = 6,\ T_4 = 7,\ T_5 = 12,\ T_6 = 13,\ T_7 = 19,\ T_8 = 20$ and $T_9 = 30$. The obtained verification results are shown in Table 6. We observe that the case study satisfies the specification TWTL specification θ_1, θ_5, and θ_7. However, the rest of the formulae θ_2, θ_3, θ_4, and θ_6 were unsatisfied. This means that not all the runs over $\mathcal{T}'$ satisfy those formulae. For instance, let us consider the case of θ_2. In $\mathcal{T}'$, there exists atleast at least one path, such as $(I_2 \rightarrow P_3 \rightarrow P_5 \rightarrow R_1 \rightarrow P_4 \rightarrow R_2 \rightarrow P_4 \rightarrow P_6 \rightarrow R_4 \rightarrow P_6 \rightarrow R_6 \rightarrow P_9 \rightarrow C_1) \wedge (I_1 \rightarrow P_1 \rightarrow P_4 \rightarrow R_1 \rightarrow P_5 \rightarrow R_3 \rightarrow P_5 \rightarrow P_7 \rightarrow R_5 \rightarrow P_7 \rightarrow R_6 \rightarrow P_9 \rightarrow C_2)$ over $\mathcal{T}'$ that violates θ_2. We observe that if the aforementioned path is mapped to the trace variables π_1 and π_2 respectively, the pair of traces also violate the HyperTWTL specification φ_2. This shows that a trace over $\mathcal{T}'$ that violates TWTL formula is equivalent to a pair of traces over $\mathcal{T}$ that violates the corresponding HyperTWTL formula. We also observe that the execution time for verifying the TWTL specifications $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$, and θ_7 are 16.30, 19.05, 22.24, 24.52, 25.19, 23.83, and 22.24 seconds, respectively. Similarly, the memory consumed for verifying these specifications are 15.21, 15.33, 17.25, 16.27, 16.04, 16.39, and 17.48 MB, respectively.

In addition to model checking of HyperTWTL, our proposed approach can be applied to other applications. One such application is motion planning in robotics. Let us assume a given 2-D grid representation where the green, red, grey, blue, and white colors represent the initial, goal, region of interest, obstacles, and the allowable states respectively. We formalize two planning objectives using HyperTWTL on the 2-D grid as follows.

Requirement 8 (Shortest path): The shortest path exists if there exists a trace π_2 that starts from the initial state through the region of interest and reaches the goal state before any other trace π_1. We consider the following time bounds for the synthesis of this requirement: $T_1 = 2, T_2 = 3, T_3 = 8, T_4 = 9, T_5 = 13$. This requirement can be formalized as a HyperTWTL formula as:

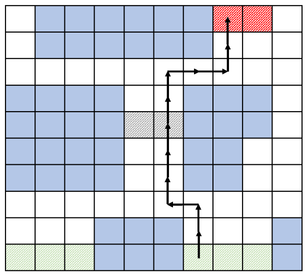

Figure 2: Strategy for shortest path φ_5

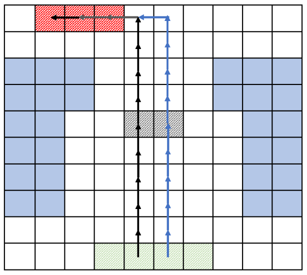

Figure 3: Strategy for initial-state opacity φ_6

$$\varphi_8 = \exists\pi_2\forall\pi_1 \cdot [\mathbf{H}^1\ s_{0_{\pi_1}} = \mathbf{H}^1\ s_{0_{\pi_2}}]^{[0,T_1]} \odot [\mathbf{H}^1\ r_{\pi_1} \wedge \mathbf{H}^1\ r_{\pi_2}]^{[T_2,T_3]} \odot ([\mathbf{H}^1\ g_{\pi_2}]^{[T_4,T_5]} \wedge [\mathbf{H}^1 g_{\pi_2}] \rightarrow [\mathbf{H}^1 g_{\pi_1}]^{[T_4,T_5]})$$

Requirement 9 (Initial-state opacity): The opacity requirement is satisfied when at least two traces π_1 and π_2 meet the following conditions : (i) both traces of the system mapped to π_1 and π_2 have the same observations but bear different secrets; and (ii) the secret of each path cannot be accurately determined by observing(partially) the system alone. For initial-state opacity requirement, let the initial state of the paths ($s_0 \in S_{init}$) be secret and observe whether both paths reach the goal states with the same set of observations. We consider the following time bounds for the synthesis of this requirement: $T_1 = 2s, T_2 = 3s, T_3 = 8s, T_4 = 9s, T_5 = 13s$. This objective can be expressed using HyperTWTL as:

$$\varphi_9 = \exists\pi_1\exists\pi_2 \cdot [\mathbf{H}^1\ s_{0_{\pi_1}} \neq \mathbf{H}^1\ s_{0_{\pi_2}}]^{[0,T_1]} \odot [\mathbf{H}^1\ r_{\pi_1} \wedge \mathbf{H}^1\ r_{\pi_2}]^{[T_2,T_3]} \odot [\mathbf{H}^1\ g_{\pi_1} \wedge \mathbf{H}^1\ g_{\pi_2}]^{[T_4,T_5]}$$

The 2-D grid environment is first converted into an equivalent TKS, and then fed to Algorithm 1 with each planning objective formalized as φ_8 and φ_9. For this, we make use of the `Synthesis()` function from [48] instead of the `Verify()` function in Algorithm 1 (Line 8) to solve the planning problem using the PyTWTL tool. For the shortest path objective, φ_8, the synthesized black path in Figure 2 is the shortest path from the initial state through to the region of interest to the goal state. In Figure 3, the synthesized path in black is the feasible strategy for the initial-state opacity objective that has the same observation as the path in blue. Both paths reach the goal state despite starting from a different initial state.

Next, we evaluate the scalability and performance of the proposed approach for path synthesis. We used varying grid sizes

Table 7: Comparison of synthesis times and memory consumed for HyperTWTL planning objectives φ_8 and φ_9

HyperTWTL Specification	Grid size	Time (Seconds)	Memory (MB)
φ_8	10×10	23.08	15.53
φ_9		17.43	15.21
φ_8	20×20	47.94	19.11
φ_9		21.63	18.96
φ_8	30×30	72.40	28.84
φ_9		28.61	23.73
φ_8	40×40	103.05	39.50
φ_9		36.72	25.35
φ_8	50×50	161.17	52.11
φ_9		56.49	31.96

ranging from 10×10 to 50×50 to synthesize paths for the formalized HyperTWTL specifications φ_8 and φ_9. We then analyze the impact of the increasing sizes on the performance of the proposed tool. The respective synthesis time and memory consumed are shown in Table 7. We observe from Table 7 that the time taken for our algorithm to synthesize a path increases with an increase in the size of the grid. For instance, the algorithm takes 23.08 seconds to synthesize a path for φ_8 on a 10×10 grid. However, while synthesizing a feasible path for the same planning objective on a grid size of 20×20, the synthesis time increases to 47.94 seconds. Similarly, the synthesis time increases to 72.40 seconds, 103.05 seconds, and 161.17 seconds while synthesizing φ_8 on 30×30, 40×40, and 50×50 grid sizes respectively. Again, while synthesizing a feasible path for φ_9 on a 10×10 grid size, our algorithm takes 17.43 seconds. The synthesis time increases to 21.63 seconds while synthesizing a feasible path for the same objective on a grid size 20×20. Once again, the synthesis time increases to 28.61 seconds, 36.72 seconds, and 56.49 seconds while synthesizing φ_8 on 30×30, 40×40, and 50×50 grid sizes respectively. Consequently, as shown in Table 7, we observe that the memory consumed by our algorithm increases with an increase in the grid size. For instance, the algorithm consumes 15.53 MB to synthesize a path for φ_8 on a 10×10 grid size. Again, while synthesizing a feasible path for the same specification on a grid size of 20×20, the memory consumed increases to 19.11 MB. Similarly, the memory consumed increases to 28.84 MB, 39.50 MB, and 52.11 seconds while synthesizing φ_8 on 30×30, 40×40 and 50×50 grid sizes respectively. Again, our algorithm consumes 15.21 MB while synthesizing a feasible path for φ_9 on a 10×10 grid size. While synthesizing a feasible path for the same objective on a grid size 20×20, the memory consumed increases to 18.96 MB. Once again, the memory consumed increases to 23.73 MB, 27.35 MB, and 31.96 MB while synthesizing φ_8 on 30×30, 40×40, and 50×50 grid sizes, respectively.

7 RELATED WORKS

HyperLTL and HyperCTL* which were first introduced in [20] extend the temporal logics LTL, CTL, and CTL* with explicit and concurrent quantifications over trace executions of a system. In recent times, multiple techniques have been proposed to monitor [1, 12, 15, 45] and verify [22, 23, 26] hyperproperties expressed as HyperLTL and HyperCTL* specifications. Similarly, other techniques have been proposed to monitor other hyper-temporal logics. HyperSTL is a bounded hyper-temporal logic for specifying hyperproperties over real-valued signals. A testing technique for verifying HyperSTL properties in cyber-physical systems is proposed in [38]. This testing technique allows for the falsification or checking of bounded hyperproperties in CPS models. HyperMTL, a hyper-temporal logic that addresses some limitations of HyperLTL in formalizing bounded hyperproperties, is proposed in [8]. In [30], the authors presented an alternate formalization and model checking approach for HyperMTL. These two works are quite similar in synchronous semantics; however, the formalization of asynchronous semantics was presented differently. While the asynchronous semantics in [11] is based on the the existence of an infinite sequence of timestamps and allows trace to proceed at different speeds, the asynchronous semantics of [30] keeps a global clock in its analysis of traces and proceeds in order. Model checking [19] has extensively been used to verify hyperproperties of models abstracted as transition systems by examining their related state transition graphs [18]. In [27], the first model checking algorithms for HyperLTL and HyperCTL* employing alternating automata were proposed, which was also adopted in [11, 30] to verify HyperMTL properties. An extensive study on the complexity of verifying hyperproperties with model checking is presented in [9]. Our formulation of HyperTWTL is closely related to the HyperMTL [11] proposed to express timed hyperproperties in discrete-time systems. However, as previously mentioned in introduction, classical TWTL formalism has several advantages including compactness obtained through the use of concatenation operator ($\odot$) which is very useful in robotic applications. A security-aware robotic motion planning approach was recently proposed using synchronous HyperTWTL specification and SMT solvers [14]. In contrast to[14], this paper presents an adequate version of HyperTWTL and addresses the model checking problem for both synchronous and asynchronous HyperTWTL.

8 CONCLUSION

This paper introduced a model checking algorithm for a hyper-temporal logic, HyperTWTL, with synchronous and asynchronous semantics. Using a Technical Surveillance Squadron (TESS) case study, we showed that HyperTWTL can express important properties related to information-flow security policies and concurrency in complex robotic systems. Our proposed model checking algorithm verifies fragments of HyperTWTL by reducing the problem to a TWTL model checking problem. In the future, we plan to propose methods and algorithms for monitoring and synthesizing the alternation-free and k-alternations fragments of HyperTWTL.

REFERENCES

[1] Shreya Agrawal and Borzoo Bonakdarpour. 2016. Runtime verification of k-safety hyperproperties in HyperLTL. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. IEEE, 239–252.

[2] Derya Aksaray, Yasin Yazıcıoğlu, and Ahmet Semi Asarkaya. 2021. Probabilistically guaranteed satisfaction of temporal logic constraints during reinforcement learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 6531–6537.

[3] Adel N Alahmadi, Saeed Ur Rehman, Husain S Alhazmi, David G Glynn, Hatoon Shoaib, and Patrick Solé. 2022. Cyber-Security Threats and Side-Channel Attacks for Digital Agriculture. *Sensors* 22, 9 (2022), 3520.

[4] Bowen Alpern and Fred B Schneider. 1985. Defining liveness. *Information processing letters* 21, 4 (1985), 181–185.

[5] Ahmet Semi Asarkaya, Derya Aksaray, and Yasin Yazicioglu. 2021. Persistent aerial monitoring under unknown stochastic dynamics in pick-up and delivery missions. In *AIAA Scitech 2021 Forum*. 1125.

[6] Ahmet Semi Asarkaya, Derya Aksaray, and Yasin Yazıcıoğlu. 2021. Temporal-logic-constrained hybrid reinforcement learning to perform optimal aerial monitoring with delivery drones. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 285–294.

[7] G. Barthe, P.R. D'Argenio, and T. Rezk. 2004. Secure information flow by self-composition. In *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004*. 100–114. https://doi.org/10.1109/CSFW.2004.1310735

[8] Borzoo Bonakdarpour, Jyotirmoy V Deshmukh, and Miroslav Pajic. 2018. Opportunities and challenges in monitoring cyber-physical systems security. In *International Symposium on Leveraging Applications of Formal Methods*. Springer, 9–18.

[9] Borzoo Bonakdarpour and Bernd Finkbeiner. 2018. The complexity of monitoring hyperproperties. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 162–174.

[10] Borzoo Bonakdarpour and Bernd Finkbeiner. 2020. Controller synthesis for hyperproperties. In *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*. IEEE, 366–379.

[11] Borzoo Bonakdarpour, Pavithra Prabhakar, and César Sánchez. 2020. Model checking timed hyperproperties in discrete-time systems. In *NASA Formal Methods Symposium*. Springer, 311–328.

[12] Borzoo Bonakdarpour, Cesar Sanchez, and Gerardo Schneider. 2018. Monitoring hyperproperties by combining static analysis and runtime verification. In *International Symposium on Leveraging Applications of Formal Methods*. Springer, 8–27.

[13] Ernest Bonnah and Khaza Anuarul Hoque. 2022. Runtime Monitoring of Time Window Temporal Logic. *IEEE Robotics and Automation Letters* 7, 3 (2022), 5888–5895.

[14] Ernest Bonnah, Luan Nguyen, and Khaza Anuarul Hoque. 2023. Motion Planning Using Hyperproperties for Time Window Temporal Logic. *IEEE Robotics and Automation Letters* 8, 8 (2023), 4386–4393. https://doi.org/10.1109/LRA.2023.3280830

[15] Noel Brett, Umair Siddique, and Borzoo Bonakdarpour. 2017. Rewriting-based runtime verification for alternation-free HyperLTL. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 77–93.

[16] Ali Tevfik Büyükkoçak, Derya Aksaray, and Yasin Yazicioglu. 2021. Distributed Planning of Multi-Agent Systems with Coupled Temporal Logic Specifications. In *AIAA Scitech 2021 Forum*. 1123.

[17] Abdullahi Chowdhury, Gour Karmakar, and Joarder Kamruzzaman. 2017. Survey of recent cyber security attacks on robotic systems and their mitigation approaches. In *Detecting and Mitigating Robotic Cyber Security Risks*. IGI global, 284–299.

[18] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. 2001. Bounded model checking using satisfiability solving. *Formal methods in system design* 19, 1 (2001), 7–34.

[19] Edmund M Clarke. 1997. Model checking. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 54–56.

[20] Michael R Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K Micinski, Markus N Rabe, and César Sánchez. 2014. Temporal logics for hyperproperties. In *International Conference on Principles of Security and Trust*. Springer, 265–284.

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

[22] Norine Coenen, Bernd Finkbeiner, César Sánchez, and Leander Tentrup. 2019. Verifying hyperliveness. In *International Conference on Computer Aided Verification*. Springer, 121–139.

[23] Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup. 2020. Synthesis from hyperproperties. *Acta informatica* 57, 1 (2020), 137–163.

[24] Bernd Finkbeiner, Christopher Hahn, and Marvin Stenger. 2017. EAHyper: satisfiability, implication, and equivalence checking of hyperproperties. In *International Conference on Computer Aided Verification*. Springer, 564–570.

[25] Bernd Finkbeiner, Christopher Hahn, and Hazem Torfah. 2018. Model checking quantitative hyperproperties. In *International Conference on Computer Aided Verification*. Springer, 144–163.

[26] Bernd Finkbeiner, Christian Müller, Helmut Seidl, and Eugen Zălinescu. 2017. Verifying security policies in multi-agent workflows with loops. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 633–645.

[27] Bernd Finkbeiner, Markus N Rabe, and César Sánchez. 2015. Algorithms for model checking HyperLTL and HyperCTL. In *International Conference on Computer Aided Verification*. Springer, 30–48.

[28] Michael R Garey. 1979. computers and intractqbility. *A Guide to the Theory of NP-Completeness* (1979).

[29] Joseph A Goguen and José Meseguer. 1982. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy*. IEEE, 11–11.

[30] Hsi-Ming Ho, Ruoyu Zhou, and Timothy M Jones. 2018. On verifying timed hyperproperties. *arXiv preprint arXiv:1812.10005* (2018).

[31] Hsi-Ming Ho, Ruoyu Zhou, and Timothy M Jones. 2021. Timed hyperproperties. *Information and Computation* 280 (2021), 104639.

[32] Tzu-Han Hsu, César Sánchez, and Borzoo Bonakdarpour. 2021. Bounded model checking for hyperproperties. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 94–112.

[33] Ron Koymans. 1990. Specifying real-time properties with metric temporal logic. *Real-time systems* 2, 4 (1990), 255–299.

[34] G Lacava, A Marotta, F Martinelli, A Saracino, A La Marra, E Gil-Uriarte, and V Mayoral Vilches. 2020. Current research issues on cyber security in robotics. (2020).

[35] Francisco J Rodríguez Lera, Camino Fernández Llamas, Ángel Manuel Guerrero, and Vicente Matellán Olivera. 2017. Cybersecurity of robotics and autonomous systems: Privacy and safety. *Robotics-legal, ethical and socioeconomic impacts* (2017).

[36] Mulong Luo, Andrew C Myers, and G Edward Suh. 2020. Stealthy tracking of autonomous vehicles with cache side channels. In *29th USENIX Security Symposium (USENIX Security 20)*. 859–876.

[37] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 152–166.

[38] Luan Viet Nguyen, James Kapinski, Xiaoqing Jin, Jyotirmoy V Deshmukh, and Taylor T Johnson. 2017. Hyperproperties of real-valued signals. In *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*. 104–113.

[39] Emmanuel Paviot-Adet, Denis Poitrenaud, Etienne Renault, and Yann Thierry-Mieg. 2022. Structural Reductions and Stutter Sensitive Properties. *arXiv preprint arXiv:2212.04218* (2022).

[40] Ryan Peterson, Ali Tevfik Buyukkocak, Derya Aksaray, and Yasin Yazıcıoğlu. 2021. Distributed safe planning for satisfying minimal temporal relaxations of TWTL specifications. *Robotics and Autonomous Systems* 142 (2021), 103801.

[41] Amir Pnueli. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. ieee, 46–57.

[42] Susan A Romano. [n. d.]. Persistent surveillance gives squadron its global purpose. https://www.af.mil/News/Article-Display/Article/1152329/persistent-surveillance-gives-squadron-its-global-purpose/

[43] Andrey Rusakov, Jiwon Shin, and Bertrand Meyer. 2014. Simple concurrency for robotics with the Roboscoop framework. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 1563–1569.

[44] Reid G Simmons. 1992. Concurrent planning and execution for autonomous robots. *IEEE Control Systems Magazine* 12, 1 (1992), 46–50.

[45] Sandro Stucki, César Sánchez, Gerardo Schneider, and Borzoo Bonakdarpour. 2019. Gray-box monitoring of hyperproperties. In *International Symposium on Formal Methods*. Springer, 406–424.

[46] Ilya Tkachev and Alessandro Abate. 2013. Formula-free finite abstractions for linear temporal verification of stochastic hybrid systems. In *Proceedings of the 16th international conference on hybrid systems: Computation and control*. 283–292.

[47] C. Vasile, D. Aksaray, and C. Belta. 2016. PyTWTL tool. Retrieved October 10, 2022 from https://sites.bu.edu/hyness/twtl/

[48] Cristian-Ioan Vasile, Derya Aksaray, and Calin Belta. 2017. Time window temporal logic. *Theoretical Computer Science* 691 (2017), 27–54.

[49] Steve Zdancewic and Andrew C Myers. 2003. Observational determinism for concurrent program security. In *16th IEEE Computer Security Foundations Workshop, 2003. Proceedings*. IEEE, 29–43.