Efficient SMT-Based Model Checking for HyperTWTL

Ernest Bonnah[1], Luan Viet Nguyen[2], and Khaza Anuarul Hoque[3]

[1] Department of Computer Engineering, Baylor University, Waco TX, USA
ernest_bonnah@baylor.edu,
[2] Department of Computer Science, University of Dayton, Dayton OH, USA
lnguyen1@udayton.edu,
[3] Department of Computer Science, University of Missouri, Columbia MO, USA
hoquek@missouri.edu

Abstract. Hyperproperties extend trace properties to express properties of sets of traces, and thus, they are increasingly popular in specifying various security and performance-related properties in domains such as autonomous, cyber-physical, and robotic systems. Specifically, Hyperproperties for time window temporal logic (HyperTWTL) are known for their compactness in specifying robotic systems' safety and security requirements. However, the existing model checking approach for HyperTWTL verification relies on automata-based model checking, which is computationally expensive and suffers from the state-space explosion problem. This paper introduces a bounded model checking approach for verifying HyperTWTL specifications using SMT solvers. Specifically, our proposed verification method reduces the HyperTWTL model checking problem to a first-order logic satisfiability problem and then uses state-of-the-art SMT solvers, i.e., Z3 and CVC4, for verification. The feasibility of the proposed HyperTWTL verification methods is demonstrated through a Technical Surveillance Squadron (TESS), a Robotic Industrial Inspection case study, and also a scalability analysis. Our results show that the proposed method can offer up to 19× speed up and 2× memory efficiency compared to the traditional automata-based model checking approach. We also show that the proposed HyperTWTL verification technique can verify large systems, whereas the traditional HyperTWTL verification method suffers from state-space explosion problem.

Keywords: Hyperproperties · Bounded Model Checking · Time Window Temporal Logic and Robotics, SMT solver.

1 Introduction

Hyperproperties [13] extend the notion of trace properties [1] from a set of traces to a set of sets of traces. This allows specifying a wide range of properties related to information-flow security [20, 32], consistency models in concurrent computing [7, 18], robustness models in cyber-physical systems [6, 19], and also service level agreements (SLA) [13]. Several types of hyperproperties and their model

checking algorithms have been proposed in the recent past, including HyperLTL [12, 14, 17, 23], HyperSTL [25], HyperMTL [8, 21], and HyperTWTL [9]. These formalisms has been successfully used to specify and verify important requirements in different domains including cyber-physical systems, robotics and machine learning. Specifically, for time-bounded and sequential tasks, HyperTWTL offers a rich expresiveness and compactness. For instance, consider a hyperproperty that requires that "*for any pair of traces π and π', A should hold for 5 time steps in trace π within the time bound* $[0, 10]$ *and B should also hold for 3 time steps in trace π' within the same time bound*". This requirement can be expressed using HyperTWTL formalism as $\varphi = \forall\pi\forall\pi' \cdot [\mathbf{H}^5\ A_\pi \wedge \mathbf{H}^3\ B_{\pi'}]^{[0,10]}$. The same requirement can be expressed as a HyperSTL formula as $\varphi = \forall\pi\forall\pi' \cdot (\mathbf{F}_{[0,10-5]}\mathbf{G}_{[0,5]}A_\pi) \wedge (\mathbf{F}_{[0,10-3]}\mathbf{G}_{[0,3]}B_{\pi'})$. In HyperMTL this requirement can be expressed as $\varphi = \forall\pi\forall\pi' \cdot \bigvee_{i=0}^{10-5}\mathbf{G}_{[i,i+5]}A_\pi \wedge \bigvee_{i=0}^{10-3}\mathbf{G}_{[i,i+3]}B_{\pi'}$.

HyperTWTL extends the classical Time Window Temporal Logic (TWTL) [30] by allowing explicit and simultaneous quantification over multiple execution traces. The classical approach for verifying HyperTWTL in [9] relies on an automata-based model checking. Traditionally, automata-based model checking is known for its high computation time, memory overhead, and may lead to a state-space explosion. Hence, we propose a more efficient and scalable approach in this paper. Specifically, we propose an SMT-based approach to verify HyperTWTL properties by converting the model checking problem to a first-order logic satisfiability problem. For example, given a HyperTWTL formula $\varphi = \forall\pi_1.\forall\pi_2 \cdot [\mathbf{H}^{15}A_{\pi_1} \wedge \mathbf{H}^{10}B_{\pi_2}]^{[0,20]}$ and a collection of Time Kripke structures (TKS) $\mathcal{M} = \langle M_1, M_2\rangle$, where M_i is an identical copy of the given Kripke structure mapped to the trace variable π_i, the process to convert both the φ and $\mathcal{M}$ to a first-order logic involves three main steps. First, we compute the unrolling bound $||\varphi||$ based on the structure of the formula. Secondly, we encode the path quantifications, initial conditions, the transition relations of each TKS M_i, and the negation of HyperTWTL formula φ as first-order logic formula represented by the encoding $[\![M_i]\!]_{||\varphi||}$ and $[\![\neg\varphi]\!]_{||\phi||}$ respectively. The combination of two encoded formulae is of the form $[\![\mathcal{M}\ \neg\varphi]\!]_{||\varphi||} = [\exists_1\pi_1] \cdot [\exists_2\pi_2] \cdot [\![M_1]\!]_{||\varphi||} \wedge [\![M_2]\!]_{||\varphi||} \wedge [\![\neg\phi]\!]_{0,||\varphi||}$. Lastly, the combined first-order logic formula unrolled to a depth of $||\varphi||$ is then solved using an off-the-shelf SMT solver. If the approach returns an affirmative answer, then the SMT solver generates a counterexample. Though the proposed approach is inspired by SMT-based bounded model checking (BMC), as earlier stated, the unrolling bound is calculated using a given HyperTWTL formula. This contrasts with the traditional BMC approach for verifying temporal logic, where an arbitrary unrolling bound is given.

To demonstrate the effectiveness of our approach, we formalize some interesting requirements of two case studies using HyperTWTL. The first case study we consider is a Technical Surveillance Squadron (TESS) [28], known for providing collaborative surveillance of designated regions to detect, identify, and locate potential nuclear explosions. In the second case study, we consider a robotic solution that automates industrial equipment inspections [2], where robots provide plant operators the information to maximize equipment uptime and improve

safety and efficiency. We use two SMT solvers, CVC4 and Z3, both known for their industrial application [4, 29], to compare their performance for verifying the HyperTWTL requirements. Finally, we compare our proposed SMT-based HyperTWTL verification performance with automata-based HyperTWTL verification. We observe that our proposed SMT-based approach offers up to 19× speed up in terms of execution speed and consumes up to 2× less memory when compared to the automata-based HyperTWTL verification approach. We also perform experiments to demonstrate the scalability of our approach and show that we can verify large robotic systems, whereas the automata-based HyperTWTL verification leads to the state-space explosion.

2 Preliminaries

Let AP be a finite set of *atomic propositions* and $\Sigma = 2^{AP}$ be the *alphabet*. We call each member of Σ an *event*. We define a timed trace t as a finite sequence of events from Σ^*, i.e., $t = (\tau_i, e_i), (\tau_{i+1}, e_{i+1}), \cdots (\tau_n, e_n) \in (\mathbb{Z}_{\geq 0} \times \Sigma)^*$ where $\tau_i \tau_{i+1} \cdots \tau_n \in \mathbb{Z}_{\geq 0}$ is a sequence of non-negative integers denoting *time-stamps* and the indices $i, n \in \mathbb{Z}_{\geq 0}$ denote *time-points*. We require $\tau_i = 0$, $\tau_i \leq \tau_{i+1}$, and for all i, $0 \leq i \leq n$. For each timed trace t, by $t[i].e$, we mean e_i and by $t[i].\tau$ we mean τ_i. We now define an indexed timed trace as a pair (t, p) where $p \in \mathbb{Z}_{\geq 0}$ is called a pointer. Indexed timed traces allow traversing a given trace by moving the pointer. Given an indexed timed trace (t, p) and $m \in \mathbb{Z}_{\geq 0}$, let $(t, p) + m$ denote the resulting trace $(t, p + m)$.

2.1 Kripke Structure

We consider timed systems modeled as timed Kripke structures with the assigned time elapse on the transitions.

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

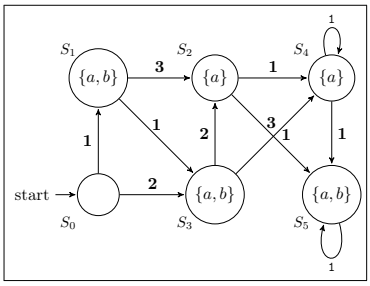

Fig. 1: Timed Kripke structure

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 $(s, 1, s) \in \delta$. An exemplary TKS is shown in Figure 1 where $S = \{S_0, S_1, S_2, S_3, S_4, S_5\}$, $S_{init} = \{S_0\}$, $\delta =$

$\{(S_0, 1, S_1), (S_0, 2, S_3), (S_1, 3, S_2), (S_1, 1, S_3), (S_2, 1, S_4), (S_2, 1, S_5), (S_3, 2, S_2), (S_3, 3, S_4), (S_4, 1, S_4), (S_4, 1, S_5), (S_5, 1, S_5)\}$, $L(S_0) = \{\}$, $L(S_1) = \{a, b\}$, $L(S_2) = \{a\}$, $L(S_3) = \{a, b\}$, $L(S_4) = \{a\}$, $L(S_5) = \{a, b\}$ and $AP = \{a, b, c, d\}$. A path over a TKS is an finite sequence of states $S_0S_1S_2 \dots S_n \in \Sigma^*$, where $S_0 \in S_{init}$ and $(S_i, d_i, S_{i+1}) \in \delta$, for each $0 \leq i < n$. A trace over TKS is of the form: $t = (\tau_0, e_0)(\tau_1, e_1)(\tau_2, e_2) \dots (\tau_n, e_n)$, such that there exists a path $S_0S_1S_2 \dots \in S^*$. Recall an event is of the form (τ_i, e_i) where $\tau_i \in \mathbb{Z}_{\geq 0}$ and $e_i = L(S_i)$.

2.2 HyperTWTL

HyperTWTL [9] is a hyper-temporal logic to specify hyperproperties for Time Window Temporal Logic (TWTL) [30] by extending TWTL with quantification over multiple and concurrent execution traces. Below we present the syntax of HyperTWTL.

Syntax of HyperTWTL: The syntax of HyperTWTL [9] is inductively defined by the grammar:

$$\varphi := \exists \pi \cdot \varphi \mid \forall \pi \cdot \varphi \mid \phi$$
$$\phi := \mathbf{H}^d a_\pi \mid \mathbf{H}^d \neg a_\pi \mid \phi_1 \wedge \phi_2 \mid \neg \phi \mid \phi_1 \odot \phi_2 \mid [\phi]^I \mid \mathbf{E}\rho \cdot \psi \mid \mathbf{A}\rho \cdot \psi$$
$$\psi := \mathbf{H}^d a_{\pi,\rho} \mid \mathbf{H}^d \neg a_{\pi,\rho} \mid \psi_1 \wedge \psi_2 \mid \neg \psi \mid \psi_1 \odot \psi_2 \mid [\psi]^{I,J}$$

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 (explained in Appendix) at a given time point. Trace quantifiers $\exists\pi$, and $\forall\pi$ are interpreted as "there exists some trace π" and "for all the traces π", respectively. Similarly, trajectory quantifiers $\mathbf{E}\rho$ and $\mathbf{A}\rho$ allow reasoning simultaneously about different trajectories. The quantifier $\mathbf{E}\rho$ means there exists at least one trajectory ρ that evaluates the relative passage of time between the traces for which the given inner temporal formula is satisfied. In contrast, $\mathbf{A}\rho$ is interpreted as all trajectories ρ satisfy the inner TWTL formula regardless of the time passage across traces. The operators $\mathbf{H}^d$, $\odot$, and $[\ ]^I$ (as well as $[\ ]^{I,J}$) represent the hold operator with $d \in \mathbb{Z}_{\geq 0}$, concatenation operator, and within operator respectively, while both I and J are discrete-time constant intervals of form $[\tau, \tau']$, 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. Given a HyperTWTL formula φ, we denote $\mathcal{V}_\varphi$ (respectively $\mathcal{P}_\varphi$) as the set of trace variables (respectively, trajectory variables) quantified in φ. Thus, we say a given formula φ is closed if for $a_{\pi,\rho}$ in φ, π and ρ are quantified in φ ($\pi \in \mathcal{V}_\varphi$ and $\rho \in \mathcal{P}_\varphi$) and no π and ρ is quantified twice in φ. 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.

Table 1: Synchronous semantics of HyperTWTL

$(\mathbb{T}, \Pi) \models \exists \pi.\varphi$	iff $\exists t \in \mathbb{T} \cdot (\mathbb{T}, \Pi[\pi \rightarrow (t,0)]) \models \varphi$
$(\mathbb{T}, \Pi) \models \forall \pi.\varphi$	iff $\forall t \in \mathbb{T} \cdot (\mathbb{T}, \Pi[\pi \rightarrow (t,0)]) \models \varphi$
$(\mathbb{T}, \Pi) \models \mathbf{H}^d a_\pi$	iff $a \in t[i].e$ for $(t,p) = \Pi(\pi)$, $\forall p \in \{i, ..., i+d\} \wedge (t[i+n].\tau - t[i].\tau) \geq d$, for some $n > 0$ and $i \geq 0$
$(\mathbb{T}, \Pi) \models \mathbf{H}^d \neg a_\pi$	iff $a \notin t[i].e$ for $(t,p) = \Pi(\pi)$, $\forall p \in \{i, ..., i+d\} \wedge (t[i+n].\tau - t[n].\tau) \geq d$, for some $n > 0$ and $i \geq 0$
$(\mathbb{T}, \Pi) \models \phi_1 \wedge \phi_2$	iff $((\mathbb{T}, \Pi) \models \phi_1) \wedge ((\mathbb{T}, \Pi) \models \phi_2)$
$(\mathbb{T}, \Pi) \models \neg\phi$	iff $\neg((\mathbb{T}, \Pi) \models \phi)$
$(\mathbb{T}, \Pi) \models \phi_1 \odot \phi_2$	iff $\exists i, j, k$ s.t. $i \leq k \leq j$ and $k = \min k' \mid i \leq k' \leq j, (\mathbb{T}_{[i,k]}, \Pi) \models \phi_1 \wedge ((\mathbb{T}_{[k+1,j]}, \Pi) \models \phi_2)$ for some $i, j \geq 0$
$(\mathbb{T}, \Pi) \models [\phi]^{[x,y]}$	iff $\exists i, j, k$ and $k \geq i + x$, s.t. $(\mathbb{T}_{[k,i+y]}, \Pi) \models \phi \wedge ((\Pi)^j - (\Pi)^{now}) \geq y$ for some $i, j \geq 0$

Semantics of HyperTWTL: The semantics of HyperTWTL [9] can be divided into *synchronous* and *asynchronous* based on the timestamps in all quantified traces that match at each point in time or proceed at different speeds, respectively. We denote the set of trace variables used in a given HyperTWTL formula φ as $\mathcal{V}_\varphi$. We define a collection of copies of TKS as $\mathcal{M} = \langle M_i \rangle_{\pi_i \in \mathcal{V}_\varphi}$, where each M_i is an identical copy of a given TKS used to represent path π_i. We therefore denote a set of traces over $\mathcal{M}$ as $\mathbb{T}$. Thus, for any given HyperTWTL formula φ, we interpret $\mathbb{T} = \langle T_{\pi_i} \rangle_{\pi_i \in \mathcal{V}_\varphi}$ as the tuple of sets of traces with a set T_{π_i} assigned to $\pi_i \in \mathcal{V}_\varphi$. Thus, for a given collection of TKS $\mathcal{M}$, we define T_{π_i} as the set of traces over the trace variable π_i coming from M_i. For any given set of sets of traces denoted as $\mathbb{T}_{[i,j]}$, we say the evaluation of all the traces in $\mathbb{T}$ against a formula starts from the time-point $i \geq 0$ up to and including the time-point $j \geq i$. Both semantics of HyperTWTL are presented below.

Synchronous Semantics of HyperTWTL: We define an assignment $\Pi : \mathcal{V} \rightarrow (\mathbb{Z}_{\geq 0} \times \Sigma)^* \times \mathbb{Z}_{\geq 0}$ as a partial function mapping trace variables to time-stamped traces. Let $\Pi(\pi) = (t,p)$ denote the time-stamped event from trace t at position p currently employed in considering trace π. We then denote the explicit mapping of the trace variable π to a trace $t \in \mathbb{T}$ at position p as $\Pi[\pi \rightarrow (t,p)]$. Thus, by $\Pi(\pi) = (t,p)$, we mean the event from the timed trace t at the position p is currently used in the analysis of trace π. Given the mapping Π, we use $(\Pi) + k$ as the k^{th} successor of Π, i.e., the k^{th} timed event of a mapped trace reached after moving k steps across Π. The hold operator $\mathbf{H}^d a_\pi$ states that the proposition a will 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 occur 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 $\phi = \phi_1 \odot \phi_2$ 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 $\phi = [\phi]^{[\tau,\tau']}$. Given Π, we define the the current instant denoted as $(\Pi)^{now}$ and

the j^{th} instant denoted as $(\Pi)^j$ as follows [8]:

$$(\Pi)^{now} = \max_{\pi \in dom(\Pi)} \{t[p].\tau \quad | \quad \text{for} \quad \Pi(\pi) = (t,p)\}$$
$$(\Pi)^{j} = \min_{\pi \in dom(\Pi)} \{t[p+j].\tau \quad | \quad \text{for} \quad \Pi(\pi) = (t,p)\}$$

We say a collection of traces $\mathbb{T}$ generated over a collection of TKS $\mathcal{M}$ satisfies a synchronous HyperTWTL formula φ if $(\mathbb{T}, \Pi) \models_s \varphi$. We present the synchronous semantics of HyperTWTL in Table 1.

Asynchronous Semantics of HyperTWTL: To define the Asynchronous semantics of HyperTWTL, we adopt the concept of trajectory as used in [23]. For a given HyperTWTL formula, a trajectory $v = v_i v_{i+1} v_{i+2} \cdots$ is a sequence of subsets of $\mathcal{P}_\varphi$, i.e. $v_i \subset \mathcal{P}_\varphi, \forall i \geq 0$. We call a trajectory a fair trajectory if, for a trace variable $\pi \in \mathcal{P}_\varphi$, there are infinitely many positions i such that $\pi \in v_i$. We denote $\mathcal{R}_\mathcal{P}$ as the set of all fair trajectories for indices from the set of trajectories $\mathcal{P}$. We now define the trajectory mapping $\Gamma : \mathcal{P}_\varphi \to \mathcal{R}_{dom(\Gamma)}$, where $dom(\Gamma) \subset \mathcal{P}_\varphi$ for which Γ is defined. We then denote the explicit mapping of the trajectory variable ρ to a trajectory v as $\Gamma[\rho \to v]$. Given (Π, Γ) where Π and Γ are the trace mapping as used in the definition of Synchronous semantics of HyperTWTL and trajectory mapping respectively, we use $(\Pi, \Gamma) + k$ as the k^{th} successor of (Π, Γ), i.e. the k^{th} reached can be reached after k steps from (Π, Γ). In defining the semantics of Asynchronous HyperTWTL, we employ the asynchronous assignment $\Pi : \mathcal{V}_\varphi \times \mathcal{P}_\varphi \to \mathbb{T} \times \mathbb{Z}_{\geq 0}$ which maps each pair of trace variable and trajectory variable, (π, ρ), into an indexed trace. Given a trace mapping Π, a trace variable π, a 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 $\Pi[(\pi, \rho) \to (t, n)]$. By $\Pi(\pi, \rho) = (t, p)$, we mean the event from the timed trace t at the position p is currently used in the analysis of trace and trajectory, π and ρ, respectively.

Let us recall that the hold operator $\mathbf{H}^d a_{\pi,\rho}$ states that the proposition a is to be repeated for d time units in trace π and trajectory ρ. Similarly $\mathbf{H}^d \neg a_{\pi,\rho}$, requires that for d time units the proposition a should not be repeated in trace π and trajectory ρ. The trace set $\mathbb{T}$ satisfies both sub-formulae in $\phi = \psi_1 \wedge \psi_2$ while in $\neg\psi$, $\mathbb{T}$, does not satisfy the given formula. A given formula with a concatenation operator in the form $\psi_1 \odot \psi_2$ 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 intended meaning of $[\phi]^{I,J}$ where $I = [\tau, \tau']$ and $J = [x, y]$ is the trace set $\mathbb{T}$ must satisfy ϕ within the time window $[\tau, \tau']$ while the difference in time elapse between any pair of traces in the set $\mathbb{T}$ must be between $[x, y]$.

For any given HyperTWTL formula φ we denote Δ as a map from $\mathcal{V}_\varphi \to \mathbb{Z}_{\geq 0}$ returns the time duration for each π in $dom(\Delta)$. We say $\Delta \in [\tau, \tau']$ whenever for all $\pi \in dom(\Delta)$, $\Delta(\pi) \in [\tau, \tau']$. Similarly, we say $\Delta \in [x, y]$ whenever for all distinct $\pi, \pi' \in dom(\Delta)$, $|\Delta(\pi') - \Delta(\pi)| \in [x, y]$. Given two indexed trace assignments Π and Π' defined within the same domain $dom(\Pi) = dom(\Pi')$, we

Table 2: 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 $a \in t[i].e$ for $(t, p) = \Pi(\pi, \rho)$, $\forall p \in \{i, ..., i+d\} \wedge$ $(t[i+n].\tau - t[i].\tau) \geq d$, for some $n > 0$ and $i < d$
$(\mathbb{T}, \Pi, \Gamma) \models_a \mathbf{H}^d \neg a_{\pi,\rho}$	iff $a \notin t[i].e$ for $(t, p) = \Pi(\pi, \rho)$, $\forall p \in \{i, ..., i+d\} \wedge$ $(t[i+n].\tau - t[i].\tau) \geq d$, for some $n > 0$ and $i < d$
$(\mathbb{T}, \Pi, \Gamma) \models_a \psi_1 \wedge \psi_2$	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 $\exists i, j, k$ s.t. $i \leq k \leq j$ and $k = \min k' \mid i \leq k' \leq j$, $((\mathbb{T}_{[i,k]}, \Pi, \Gamma) \models_a \psi_1), \wedge ((\mathbb{T}_{[k+1,j]}, \Pi, \Gamma) \models_a \psi_2)$ for some $i, j \geq 0$
$(\mathbb{T}, \Pi, \Gamma) \models_a [\psi]^{[\tau,\tau'],[x,y]}$	iff $\exists i, j, k$ s.t. $k \geq i + \tau$, $(\mathbb{T}_{[k,i+\tau']}, \Pi, \Gamma) \models_a \psi \wedge$ $\lvert\Delta((\Pi + j) - \Pi)\rvert \in [\tau, \tau'] \wedge \lvert\Delta^j((\Pi, \Gamma), (\Pi', \Gamma'))\rvert \in [x, y]$, for some $i, j > 0$

denote $\Delta(\Pi, \Pi')(\pi)$ as the map from $\mathcal{V}_\varphi \rightarrow \mathbb{Z}_{\geq 0}$ that returns the time duration for each trace assignment as $\Delta(\Pi, \Pi')(\pi) = (\Pi'(\pi)).\tau - (\Pi(\pi)).\tau$. Likewise, given two distinct indexed trace and trajectory assignments (Π, Γ) and (Π', Γ') of the same domain, we denote $\Delta^j((\Pi, \Gamma), (\Pi', \Gamma))$ as the duration of time that elapses from the current evaluation instant to the evaluation instance obtained after j steps. This is defined formally as $\Delta^j((\Pi, \Gamma), (\Pi', \Gamma')) = \Delta(\Pi, \Pi')(\pi)$, where $(\Pi', \Gamma') = (\Pi, \Gamma)^j$. Now, we denote the satisfaction of asynchronous semantics of HyperTWTL formula φ over trace mapping Π, trajectory mapping Γ, and a set of traces $\mathbb{T}$ as $(\mathbb{T}, \Pi, \Gamma) \models_a \varphi$. The asynchronous semantics of HyperTWTL is presented in Table 2.

3 SMT-based Model Checking for HyperTWTL

Given a collection of TKS $\mathcal{M}$, a HyperTWTL formula φ, and an unrolling bound $||\varphi||$ (discussed in the next section), the model checking problem is to determine whether $\mathcal{M} \models \varphi$. We assume that the input formula φ has been converted into a negation-normal form (NNF) denoted as $\neg\varphi$. The model checking approach takes as an input NNF of the HyperTWTL formula $\neg\varphi$ and TKS $\mathcal{M}$. Let us recall from Section 2 that φ can be either a synchronous or an asynchronous HyperTWTL formula. In the latter case, we need to translate the asynchronous HyperTWTL formula to an equivalent synchronous HyperTWTL formula. To achieve this, we first generate a set of invariant traces $inv(\mathbb{T})$ from a trace set $\mathbb{T}$ generated over the TKS $\mathcal{M}$. We then construct an equivalent synchronous formula φ_s from the asynchronous formula φ_s such that $\mathbb{T} \models_a \varphi_a$ if and only if $inv(\mathbb{T}) \models_s \varphi_s$. For more details on this approach of converting an asynchronous HyperTWTL formula to an equivalent synchronous HyperTWTL formula, we refer the readers to the Appendix. Next, the TKS $\mathcal{M}$ and NNF of the HyperTWTL formula $\neg\varphi$ are fed into an SMT encoder to generate a first-order logic

formula of the form $[\![\mathcal{M}, \neg\varphi]\!]_{||\varphi||}$ by encoding the initial condition, the transition relations, and unrolling $\mathcal{M}$ and $\neg\varphi$ to a depth of $||\varphi||$. Finally, we utilize off-the-shelf SMT solvers to solve the first order logic formula $[\![\mathcal{M}, \neg\varphi]\!]_{||\varphi||}$ and determine if $\mathcal{M} \models \neg\varphi$. If the SMT returns true, then a counterexample has been identified, otherwise, $\mathcal{M} \models \varphi$ holds.

3.1 Calculating Unrolling Bound from HyperTWTL

The satisfaction of a HyperTWTL formula can be decided within a fixed time bound. Let $||\varphi||$ denote the maximum time needed to satisfy the HyperTWTL formula φ and it can be computed as follows:

$$||\varphi|| = \begin{cases} ||\varphi|| & \text{if} \quad \varphi \in \{\exists\pi \cdot \varphi, \forall\pi \cdot \varphi\} \\ d & \text{if} \quad \varphi \in \{\mathbf{H}^d a_\pi, \mathbf{H}^d \neg a_\pi\} \\ max(||\phi_1||, ||\phi_2||) & \text{if} \quad \varphi \in \{\phi_1 \wedge \phi_2, \phi_1 \vee \phi_2\} \\ ||\phi|| & \text{if} \quad \varphi = \neg\phi \\ ||\phi_1|| + ||\phi_2|| + 1 & \text{if} \quad \varphi = \phi_1 \odot \phi_2 \\ \tau' & \text{if} \quad \varphi \in \{[\phi_1]^{[\tau,\tau']}\} \end{cases} \tag{1}$$

We use the computed deadline $||\varphi||$ as the *unrolling bound* to determine the satisfiability of a HyperTWTL formula. Note, this contrasts the traditional BMC techniques which uses a given arbitrary unrolling bound.

Example 1. Let us consider a HyperTWTL formula φ as follows.

$$\varphi_1 = \forall\pi_1 \exists\pi_2 \cdot [\mathbf{H}^2 a_{\pi_1} \wedge \mathbf{H}^2 a_{\pi_2}]^{[0,2]} \odot [\mathbf{H}^2 a_{\pi_1} \vee \mathbf{H}^2 b_{\pi_2}]^{[3,7]} \tag{2}$$

Using Equation (1), we can calculate $||\varphi_1|| = 10$ time units.

3.2 Encoding the TKS

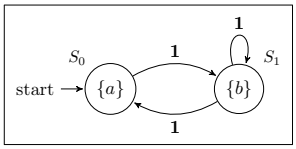

Fig. 2: TKS M

The encoding of a collection of TKS $\mathcal{M}$ upto bound $||\varphi||$ into a first-order logic formula is inspired by the BMC encoding of LTL [3]. Intuitively, the states of the $\mathcal{M}$ are represented by a set of variables S. Let S_i be new copies of S, where $i \in [0, ||\varphi||]$ which captures the evolution of states over time. Consider the HyperTWTL formula φ_1 in Equation (2) above, we use two identical copies of a given TKS to represent different paths π_1 and π_2 on the TKS, denoted as M_1 and M_2, i.e. $\mathcal{M} = \langle M_1, M_2 \rangle$. Therefore, for each copy M_i, we unroll the transition relation $[\![M_i]\!]_{||\varphi_1||}$ as follows.

$$[\![M_i]\!]_{||\varphi_1||} = I(S_0) \wedge \bigwedge_{i=0}^{||\varphi_1||-1} R(S_i, S_{i+1}) \tag{3}$$

In Equation (3), $I(S_0)$ is the characteristic function that encodes the initial states and $R(S_i, S_{i+1})$ is the function that encodes transition relation for states in S_i and their successor states in S_{i+1} between time steps i and $i+1$.

Table 3: Encoding the inner TWTL formula

$[\![\mathbf{H}^d\ a_\pi]\!]_{i,\|\|\varphi\|\|}$	$:= [\![\mathbf{H}^d a_\pi]\!]_i\ \forall i \leq \|\|\varphi\|\|$
$[\![\mathbf{H}^d\ \neg a_\pi]\!]_{i,\|\|\varphi\|\|}$	$:= [\![\mathbf{H}^d \neg a_\pi]\!]_i\ \ \forall i \leq \|\|\varphi\|\|$
$[\![\phi_1 \wedge \phi_2]\!]_{i,\|\|\varphi\|\|}$	$:= [\![\phi_1]\!]_{i,\|\|\varphi\|\|} \wedge [\![\phi_2]\!]_{i,\|\|\varphi\|\|},\ \ \forall i \leq \|\|\varphi\|\|$
$[\![\neg\phi]\!]_{i,\|\|\varphi\|\|}$	$:= \neg[\![\phi]\!]_{i,\|\|\varphi\|\|},\ \ \forall i \leq \|\|\varphi\|\|$
$[\![\phi_1 \odot \phi_2]\!]_{i,\|\|\varphi\|\|}$	$:= \exists k = \arg\min_{i \leq k \leq \|\|\varphi\|\|} [\![\phi_1]\!]_{i,k} \wedge$ $[\![\phi_2]\!]_{k+1,\|\|\varphi\|\|}\ \forall i \leq \|\|\varphi\|\|$
$[\![[\phi]^{[\tau,\tau']}]\!]_{i,\|\|\varphi\|\|}$	$:= \exists k \geq i+\tau, s.t. [\![\phi]\!]_{k,i+\tau} \wedge (\|\|\varphi\|\| - i \geq \tau'),$ $\forall i \leq \|\|\varphi\|\|$

Example 2. Consider the Kripke structure in Figure 2 and a HyperTWTL formula

$$\varphi_2 = \forall\pi_1 \forall\pi_2 \cdot [\mathbf{H}^3 a_{\pi_1} \wedge \mathbf{H}^3 b_{\pi_2}]^{[0,3]} \tag{4}$$

For a bound $||\varphi_2|| = 3$, we unroll the transition relation for copy M_1 as follows.

$$[\![M_1]\!]_{||3||} = I(S_0) \wedge R(S_0, S_1) \wedge R(S_1, S_2) \wedge R(S_2, S_3) \tag{5}$$

3.3 Encoding the inner TWTL Formula

Let φ be a HyperTWTL formula of the form $\varphi = Q_1\pi_1 \ldots Q_n\pi_n \cdot \phi$ where each $Q_j \in \{\forall, \exists\}$ $(j \in [1, n])$ and ϕ is the inner TWTL formula. For each $j \in [1, n]$, the path quantification $Q_j\pi_j$ is represented by

$$[Q_j\pi_j] = Q_j S_0 Q_j S_1 \cdots Q_j S_{||\varphi||-1} \tag{6}$$

Given the negated formula $\neg\varphi$, we unroll the TWTL formula on a path π, with bound $||\varphi||$ resulting in a first-order logic formula which can be inductively defined in Table 3.

Example 3. Consider the HyperTWTL formula φ_2 in Equation (4) above. The negation of Equation (4) (refer Theorem 1) can be expressed as follows.

$$\neg\varphi_2 = \exists\pi_1 \exists\pi_2 \cdot \underbrace{[\mathbf{H}^3 \neg a_{\pi_1} \vee \mathbf{H}^3 \neg b_{\pi_2}]^{[0,3]}}_{\neg\phi} \tag{7}$$

From the structure of Equation (7), the inner TWTL formula $\neg\phi$ is given as $\neg\phi = [\mathbf{H}^3 \neg a_{\pi_1} \vee \mathbf{H}^3 \neg b_{\pi_2}]^{[0,3]}$. Based on Table 3, unrolling $\neg\phi$ with a computed bound $||\neg\varphi_2|| = 3$ can be expressed as follows.

$$\begin{aligned}[\![\neg\phi]\!]_{0,[3]} = [\mathbf{H}^3 \neg a_{\pi_1} \vee \mathbf{H}^3 \neg b_{\pi_2}]_0^{[0,3]} \wedge [\mathbf{H}^3 \neg a_{\pi_1} \vee \mathbf{H}^3 \neg b_{\pi_2}]_1^{[0,3]} \\ \wedge\ [\mathbf{H}^3 \neg a_{\pi_1} \vee \mathbf{H}^3 \neg b_{\pi_2}]_2^{[0,3]} \wedge\ [\mathbf{H}^3 \neg a_{\pi_1} \vee \mathbf{H}^3 \neg b_{\pi_2}]_3^{[0,3]}\end{aligned} \tag{8}$$

3.4 Combining the Encodings

Given a HyperTWTL formula of the form $\varphi = Q_1\pi_1 \dots Q_n\pi_n \cdot \phi$ and a collection of TKS $\mathcal{M} = \langle M_1, \cdots, M_n \rangle$, the verification problem of HyperTWTL specifications can be formulated by constructing the first-order logic formula $[\![\mathcal{M}, \neg\varphi]\!]_{||\varphi||}$ as follows.

$$[\![\mathcal{M}, \neg\varphi]\!]_{||\varphi||} = [Q_1\pi_1] \dots [Q_n\pi_n] \cdot [\![M_1]\!]_{||\varphi||} \, \square_1 \, \dots [\![M_n]\!]_{||\varphi||} \, \square_n \, [\![\neg\phi]\!]_{0,||\varphi||} \tag{9}$$

where $[Q_j\pi_j]$ for $j \in [1, n]$ is defined in (6), $[\![M_j]\!]_{||\varphi||}$ for $j \in [1, n]$ is defined in (3), $\square_i = \wedge$ if $Q_i = \exists$, and $\square_i = \rightarrow$ if $Q_i = \forall$, for $i \in \mathcal{V}_\varphi$ and $\neg\phi$ is the negated inner TWTL formula ϕ of the HyperTWTL formula φ.

Example 4. Let us consider the Kripke structure in Fig. 1 and the HyperTWTL formula $\varphi = \forall\pi_1 \forall\pi_2 \cdot [\mathbf{H}^3 a_{\pi_1} \wedge \mathbf{H}^3 b_{\pi_2}]^{[0,3]}$ with $||\varphi|| = 3$. Let $\mathcal{M} = \langle M_1, M_2 \rangle$ denote identical collection of the Kripke structure representing paths π_1 and π_2 respectively. The resulting combined first-order logic formula to be solved is given as follows.

$$[\![\mathcal{M}, \neg\varphi]\!]_3 = [\exists_1\pi_1] \cdot [\exists_2\pi_2] \cdot [\![M_1]\!]_3 \wedge [\![M_2]\!]_3 \wedge [\![\neg\phi]\!]_{0,3} \tag{10}$$

Theorem 1. Given a collection TKS $\mathcal{M}$, a HyperTWTL formula φ with an unrolling bound of $||\varphi||$ and sets of traces $\mathbb{T}$ over $\mathcal{M}$, if $[\![\mathcal{M}, \neg\varphi]\!]_{||\varphi||}$ is satisfiable, i.e. $(\mathbb{T}, \Pi) \not\models_s \varphi$, then $\mathcal{M} \not\models_s \varphi$.

4 Experimental Results

To demonstrate the effectiveness of our approach, we consider two case studies and compare their performance with the automata-based HyperTWTL model checking approach [9]. We present the details of these case studies and the obtained results in the following sections.

4.1 Case Study I: Autonomous Security Robots

Our Case Study-1 resembles a security patrol within a community with multiple autonomous security robots [24]. In this case study, the autonomous security robots are augmented with intelligent video surveillance systems that move along patrol routes to different areas while identifying potential intruders, incidents, crimes, etc., and relaying information to an operator in a remote base station for data processing. Let us consider an environment to be patrolled in Figure 3 which is composed of 2 initial positions I_1 and I_2, 2 charging stations C_1 and C_2, 12 allowable states $P_1, \dots, P_{12}$ and 4 regions of interest to be patroled R_1 to R_4. Each patrol starts from any of the initial states (grey) and subsequently proceeds along the patrol routes through the allowable states (white). On each patrol, it is required each security robot surveils all regions of interest (blue) before proceeding to any of the charging stations (yellow). We abstract the patrol environment into

Table 4: Requirements expressed in HyperTWTL in Case Study I

No.	Description	HyperTWTL Specification
1	Mutation Testing	$\varphi_1 = \exists\pi_1 \forall\pi_2 \cdot\ [\mathbf{H}^d\ t^m_{\pi_1}\ \wedge\ \mathbf{H}^d\ t^{\neg m}_{\pi_2}]^{[0,T_9]} \wedge [\mathbf{H}^1\ I_{\pi_1}\ =\ \mathbf{H}^1\ I_{\pi_2}]^{[0,T_1]} \odot$ $[\mathbf{H}^1\ R_{1\pi_1}\ \wedge\ \mathbf{H}^1\ R_{1\pi_2}]^{[T_2,T_3]} \odot\ [\mathbf{H}^1\ R_{2\pi_1}\ \wedge\ \mathbf{H}^1\ R_{2\pi_1}]^{[T_4,T_5]} \odot$ $[\mathbf{H}^1\ R_{3\pi_2}\ \wedge\ \mathbf{H}^1\ R_{3\pi_2}]^{[T_6,T_7]} \odot [\mathbf{H}^1\ R_{4\pi_1}\ \wedge\ \mathbf{H}^1\ R_{4\pi_2}]^{[T_8,T_9]} \odot$ $[\mathbf{H}^1\ C_{\pi_1}\ \neq\ \mathbf{H}^1\ C_{\pi_2}]^{[T_{10},T_{11}]}$, where $d = T9$
2	Opacity	$\varphi_2 = \exists\pi_1 \exists\pi_2 \cdot\ [\mathbf{H}^1\ I_{\pi_1}\ \wedge\ \mathbf{H}^1\ I_{\pi_2}]^{[0,T_1]} \odot\ [\mathbf{H}^1\ R_{1\pi_1}\ \wedge\ \mathbf{H}^1\ R_{1\pi_2}]^{[T_2,T_3]}$ $\odot\ [\mathbf{H}^1\ R_{2\pi_1}\ \wedge\ \mathbf{H}^1\ R_{2\pi_1}]^{[T_4,T_5]} \odot [\mathbf{H}^1\ R_{3\pi_2}\ \wedge\ \mathbf{H}^1\ R_{3\pi_2}]^{[T_6,T_7]}$ $\odot\ [\mathbf{H}^1\ R_{4\pi_1}\ \wedge\ \mathbf{H}^1\ R_{4\pi_2}]^{[T_8,T_9]}) \odot\ [\mathbf{H}^1\ C_{\pi_1}\ \wedge\ \mathbf{H}^1\ C_{\pi_2}]^{[T_10,T_11]}$
3	Side-Channel Timing Attacks	$\varphi_3 = \forall\pi_1 \forall\pi_2 \cdot \mathbf{A}\rho\mathbf{E}\rho' \cdot\ [\mathbf{H}^1\ I_{\pi_1,\rho}\ \wedge\ \mathbf{H}^1\ I_{\pi_2,\rho'}]^{[0,T_1]}\ \rightarrow\ [\mathbf{H}^1\ R_{1\pi_1,\rho} \wedge$ $\mathbf{H}^1\ R_{1\pi_2,\rho'}]^{[T_2,T_3]} \odot\ [\mathbf{H}^1\ R_{2\pi_1,\rho} \wedge\ \mathbf{H}^1\ R_{2\pi_1,\rho}]^{[T_4,T_5]} \odot [\mathbf{H}^1\ R_{3\pi_2,\rho'} \wedge$ $\mathbf{H}^1\ R_{3\pi_2,\rho'}]^{[T_6,T_7]} \odot [\mathbf{H}^1\ R_{4\pi_1,\rho} \wedge\ \mathbf{H}^1\ R_{4\pi_2,\rho'}]^{[T_8,T_9]} \odot\ [\mathbf{H}^1\ C_{\pi_1,\rho} \wedge$ $\mathbf{H}^1\ C_{\pi_2,\rho'}]^{[T_{10},T_{11}]}$
4	Non-Interference	$\varphi_4 = \forall\pi_1 \exists\pi_2 \cdot \mathbf{A}\rho \cdot\ [\mathbf{H}^1\ I_{\pi_1,\rho}\ \neq\ \mathbf{H}^1\ I_{\pi_2,\rho}]^{[0,T_1]}\ \rightarrow\ [\mathbf{H}^1\ R_{1\pi_1,\rho} \wedge$ $\mathbf{H}^1\ R_{1\pi_2,\rho}]^{[T_2,T_3]} \odot\ [\mathbf{H}^1\ R_{2\pi_1,\rho} \wedge\ \mathbf{H}^1\ R_{2\pi_1,\rho}]^{[T_4,T_5]} \odot [\mathbf{H}^1\ R_{3\pi_2,\rho} \wedge$ $\mathbf{H}^1\ R_{3\pi_2,\rho}]^{[T_6,T_7]} \odot [\mathbf{H}^1\ R_{4\pi_1,\rho} \wedge\ \mathbf{H}^1\ R_{4\pi_2,\rho}]^{[T_8,T_9]} \odot\ [\mathbf{H}^1\ C_{\pi_1,\rho}\ =$ $\mathbf{H}^1\ C_{\pi_2,\rho}]^{[T_{10},T_{11}]}$

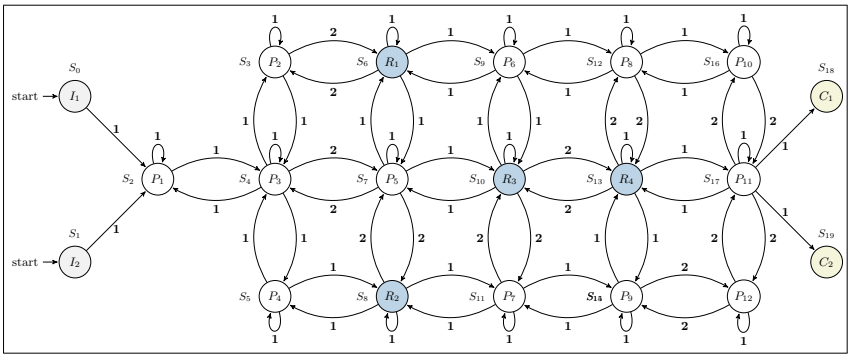

Fig. 3: The Patrol environment

a weighted graph where the nodes represent the initial states, charging stations, allowable states, and regions of interest, while the edges represent transitions between the nodes, and the assigned weights represent travel times associated with the transitions. We further abstract the motion of each security robot into a transition system derived from the patrol environment by splitting all transitions to have an edge weight of 1 time unit. Based on this case study, we consider 4 different scenarios with 4 different HyperTWTL specifications, including mutation testing, opacity, side-channel attacks and non-interference. We formalize these requirements in HyperTWTL as follows. Note, in φ_2 and all subsequent formulae, "=" is not an arithmetic operator but a notation of simplification such that $[\mathbf{H}^1\ I_{\pi_1} = \mathbf{H}^1\ I_{\pi_2}]$ stands for $\bigwedge_{i\in I}([\mathbf{H}^1\ i_{\pi_1} \wedge \mathbf{H}^1\ i_{\pi_2}])$.

Requirement 1 (Mutation testing): An interesting application of hyperproperty 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

Table 5: Verification results of HyperTWTL properties for Case Study I

HyperTWTL Req.	Description	Verdict	Z3		CVC4		AMC	
			Time (Seconds)	Memory (MB)	Time (Seconds)	Memory (MB)	Time (Seconds)	Memory (MB)
φ_1	Mutation Testing	SAT	1.450	8.604	0.844	10.554	16.103	16.893
φ_2	Opacity	UNSAT	1.453	8.691	0.892	10.564	15.952	17.110
φ_3	Side-Channel Timing Attacks	SAT	1.427	8.613	0.839	10.415	16.186	16.950
φ_4	Non-Interference	SAT	1.421	8.611	0.885	10.425	16.225	16.832

Table 6: Verification time for HyperTWTL properties for Case Study I

HyperTWTL properties	Z3 (Seconds)				CVC4 (Seconds)				AMC (Seconds)			
	Unrolling bounds ($\|\|\varphi\|\|$)				Unrolling bounds ($\|\|\varphi\|\|$)				Unrolling bounds ($\|\|\varphi\|\|$)			
	51	75	100	125	51	75	100	125	51	75	100	125
φ_1	1.450	2.588	3.629	4.741	0.844	1.775	2.772	3.682	16.103	17.165	18.386	19.507
φ_2	1.453	2.541	3.652	4.782	0.892	1.744	2.628	3.671	15.952	16.285	17.733	18.895
φ_3	1.427	2.573	3.681	4.794	0.839	1.710	2.631	3.766	16.189	17.085	18.386	19.297
φ_4	1.421	2.595	3.648	4.766	0.885	1.725	2.711	3.729	16.225	17.198	18.738	19.098

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 synchronous HyperTWTL formula φ_1 as shown in Table 4.

Requirement 2 (Opacity): Information-flow security policies define what users can learn about a system while (partially) observing the system. A system is said to be opaque if it meets two requirements: (i) there exist at least two executions of the system mapped to π_1 and π_2 with the same observations but bearing a distinct secret, and (ii) the secret of each path cannot be accurately determined only by observing the system. For example, let the surveillance route be secret, and the initial state I_1 or I_2 be the only information a system user can observe. This can be formalized as a synchronous HyperTWTL formula φ_2 as shown in Table 4.

Requirement 3 (Side-channel timing attacks): A side-channel timing attack is a security threat that attempts to acquire sensitive information from the surveillance mission by exploiting the execution time of the mission. Let us assume two security robots start from any initial states I_1 or I_2 simultaneously, and their executions are mapped to π_1 and π_2, respectively. To design a countermeasure against this attack, it is required that for any pair of executions, if both robots start from any initial states simultaneously, they should reach the charging state C_1 or C_2 within close enough time after finishing their surveillance tasks. This can be formalized as an asynchronous HyperTWTL formula φ_3 as shown in Table 4.

Table 7: Memory consumption for HyperTWTL verification for Case Study I

HyperTWTL properties	Z3 (MB)				CVC4 (MB)				AMC (MB)			
	Unrolling bounds ($\|\|\varphi\|\|$)				Unrolling bounds ($\|\|\varphi\|\|$)				Unrolling bounds ($\|\|\varphi\|\|$)			
	51	75	100	125	51	75	100	125	51	75	100	125
φ_1	8.604	13.965	20.091	24.214	10.554	16.117	23.343	27.868	17.933	19.573	26.304	32.830
φ_2	8.691	13.883	20.082	24.253	10.564	16.065	22.957	27.748	16.057	18.463	25.457	31.487
φ_3	8.613	13.834	20.051	24.269	10.415	16.219	23.117	27.445	17.578	19.931	26.647	31.608
φ_4	8.611	13.840	20.067	24.283	10.425	16.269	23.002	27.678	17.711	19.156	26.483	31.372

Requirement 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. For a set of traces, let us assume that the initial state I_1 or I_2 is a high variable (high security) and paths from initial states to charging states C_1 or C_2 through $R_1, \ldots, R_6$ denote low variable (low security). The surveillance system satisfies non-interference if, for all executions, there exists another execution that starts from a different high variable (i.e., the initial states are different), and at the end of the mission, they are in the same low variable states (i.e., charging states C_1 or C_2 are the same). This can be formalized as an asynchronous HyperTWTL formula φ_4 as shown in Table 4.

4.2 Case Study 1: Experimental Results

The conversion from the TKS and the HyperTWTL specifications to first-order logic expressions (resembling Equation (9)) is implemented in Python 3.7. The obtained first-order logic formula is then fed to Z3 and CVC4 SMT solvers for verification on a Windows 10 system with 64 GB RAM and Intel Core(TM) i9-10900 CPU (3.70 GHz). Z3 and CVC4 are widely known for their industrial applications [4,29]. The following time bounds are considered for the verification of all the HyperTWTL properties in Table 4: $T_1 = 1$, $T_2 = 2$, $T_3 = 3$, $T_4 = 4$, $T_5 = 7$, $T_6 = 8$, $T_7 = 9$, $T_8 = 10$, $T_9 = 12$, $T_{10} = 13$ and $T_{13} = 15$. Since the time bounds are the same for HyperTWTL formulae from φ_1–φ_4, their unrolling bound is also the same, i.e., $||\varphi|| = 51$.

In the first set of experiments, we verify the HyperTWTL specifications using the $Z3$ and CVC4 SMT solvers and compare them with the automata-based model checking (AMC) approach in [9]. The obtained results are shown in Table 5. Note, since φ_3 and φ_4 are asynchronous HyperTWTL specifications, we convert them to equivalent synchronous HyperTWTL specifications following the method described in [9] before running these experiments. We observe that specifications φ_1, φ_3 and φ_4 were satisfied using both Z3 and CVC4 as well as the AMC approach, whereas φ_2 was unsatisfied. We also observe that the verification time of the HyperTWTL formulae never exceeded 1.453 seconds in Z3 and 0.892 seconds in CVC4. In contrast, it took up to 16.225 seconds in the AMC approach to verify these properties. This shows a 11× and 19× speed up for Z3 and CVC4, respectively, regarding execution time. Regarding the memory

Table 8: Requirements expressed in HyperTWTL in Case Study II

No.	Description	HyperTWTL Specification
5	Shortest Path	$\varphi_5 = \exists\pi_1 \forall\pi_2.\ [\mathbf{H}^1\ S_{\pi_1}\ \wedge\ \mathbf{H}^1\ S_{\pi_2}]^{[0,T_1]}\ \odot\ [\mathbf{H}^1\ E_{1\pi_1}\ \wedge\ \mathbf{H}^1\ E_{1\pi_2}]^{[T_2,T_3]}\ \odot\ [\mathbf{H}^1\ E_{2\pi_1}\ \wedge\ \mathbf{H}^1\ E_{2\pi_2}]^{[T_4,T_5]}\ \odot\ [\mathbf{H}^1\ E_{3\pi_1}\ \wedge\ \mathbf{H}^1\ E_{3\pi_2}]^{[T_6,T_7]}\ \odot\ [\mathbf{H}^1\ E_{4\pi_1}\ \wedge\ \mathbf{H}^1\ E_{4\pi_2}]^{[T_8,T_9]}\ \odot\ [\mathbf{H}^1\ E_{5\pi_1}\ \wedge\ \mathbf{H}^1\ E_{5\pi_2}]^{[T_{10},T_{11}]}\ \odot\ [\mathbf{H}^1\ E_{6\pi_1}\ \wedge\ \mathbf{H}^1\ E_{6\pi_2}]^{[T_{12},T_{13}]}\ \odot\ ([\mathbf{H}^1\ L_{\pi_2}]^{[T_{14},T_{15}]}\ \wedge\ [\mathbf{H}^1\ L_{\pi_1}] \rightarrow [\mathbf{H}^1\ L_{\pi_2}]^{[T_{14},T_{15}]})$
6	Symmetry	$\varphi_6 = \exists\pi_1 \forall\pi_2 \cdot\ [\mathbf{H}^1\ S_{\pi_1}\ \wedge\ \mathbf{H}^1\ S_{\pi_2}]^{[0,T_1]}\ \odot\ [\mathbf{H}^1\ E_{1\pi_1}\ \wedge\ \mathbf{H}^1\ E_{1\pi_2}]^{[T_2,T_3]}\ \odot\ [\mathbf{H}^1\ E_{2\pi_1}\ \wedge\ \mathbf{H}^1\ E_{3\pi_1}]^{[T_4,T_7]}\ \odot\ [\mathbf{H}^1\ E_{4\pi_1}\ \wedge\ \mathbf{H}^1\ E_{5\pi_2}]^{[T_6,T_{11}]}\ \odot\ [\mathbf{H}^1\ E_{6\pi_1}\ \wedge\ \mathbf{H}^1\ E_{6\pi_2}]^{[T_{12},T_{13}]} \odot\ [\mathbf{H}^1\ L_{\pi_1}\ \wedge\ \mathbf{H}^1\ L_{\pi_2}]^{[T_{14},T_{15}]}$
7	Linearizability	$\varphi_7 = \forall\pi_1 \exists\pi_2 \cdot\ [\mathbf{H}^1\ S_{\pi_1}\ =\ \mathbf{H}^1\ S_{\pi_2}]^{[0,T_1]}\ \odot\ [\mathbf{H}^1\ E_{1\pi_1}\ \wedge\ \mathbf{H}^1\ E_{1\pi_2}]^{[T_2,T_3]}\ \odot\ [\mathbf{H}^1\ E_{2\pi_1}\ \wedge\ \mathbf{H}^1\ E_{2\pi_2}]^{[T_4,T_5]}\ \odot\ [\mathbf{H}^1\ E_{3\pi_1}\ \wedge\ \mathbf{H}^1\ E_{3\pi_2}]^{[T_6,T_7]}\ \odot\ [\mathbf{H}^1\ E_{4\pi_1}\ \wedge\ \mathbf{H}^1\ E_{4\pi_2}]^{[T_8,T_9]}\ \odot\ [\mathbf{H}^1\ E_{5\pi_1}\ \wedge\ \mathbf{H}^1\ E_{5\pi_2}]^{[T_{10},T_{11}]}\ \odot\ [\mathbf{H}^1\ E_{6\pi_1}\ \wedge\ \mathbf{H}^1\ E_{6\pi_2}]^{[T_{12},T_{13}]}\ \odot\ [\mathbf{H}^1\ L_{\pi_1}\ =\ \mathbf{H}^1\ L_{\pi_2}]^{[T_{14},T_{15}]}$

consumption, verifying these specifications using Z3 and CVC4 never exceeded 8.691 MB and 10.564 MB, respectively. In contrast, AMC consumed up to 17.110 MB of memory. This shows that our SMT-based verification approach is 2× and 1.6× more memory-efficient while using Z3 and CVC4, respectively.

In the second set of experiments, we evaluate the performance of Z3 and CVC4 solvers for verifying HyperTWTL properties against different unrolling bounds, i.e., analyze the impact of $||\varphi||$ on the verification performance. For this, we vary the $||\varphi||$ in the range of 51 to 125 for φ_1–φ_4 and record their respective verification time and memory as shown in Tables 6 and 7. Table 6 shows that CVC4 is faster than Z3 and the AMC approach regarding verification time. For instance, while verifying φ_1 for $||\varphi|| = 125$, Z3 and AMC approach take 4.741 seconds and 19.507 seconds respectively. In contrast, verifying the same property for $||\varphi|| = 125$ using CVC4 takes only 3.682 seconds. This shows that CVC4 is faster than both the Z3 and AMC approaches. Similar pattern is observed while verifying φ_4 for $||\varphi|| = 75$. Z3 and AMC approaches take 2.595 seconds and 18.738 seconds, respectively, whereas CVC4 takes only 1.724 seconds. A similar trend is also observed for the rest of the HyperTWTL properties. We also observe that execution time increases linearly for verifying HyperTWTL properties against increasing unrolling bounds, irrespective of the techniques used for verification.

Consequently, as shown in Table 7, we observe that Z3 consumes less memory than CVC4 and the AMC for verifying the HyperTWTL properties. For instance, while verifying φ_2 for $||\varphi|| = 100$, CVC4 and AMC consume 22.957 MB and 25.457 MB in memory, respectively. In contrast, verifying the same property for $||\varphi|| = 100$ using Z3 takes 20.082 MB. This shows that Z3 is more memory efficient than CVC4 and, of course, AMC. Similarly, while verifying φ_4 for $||\varphi|| = 51$, CVC4 and the AMC consume 10.425 MB and 17.711 MB in memory, respectively, whereas Z3 consumes only 8.611 MB. A similar trend of memory consumption is observed for the rest of the HyperTWTL specifications. Indeed, we also observe a linear trend in memory consumption with increasing HyperTWTL unrolling bound irrespective of the verification method used.

4.3 Case Study II: Industrial Inspection Robots

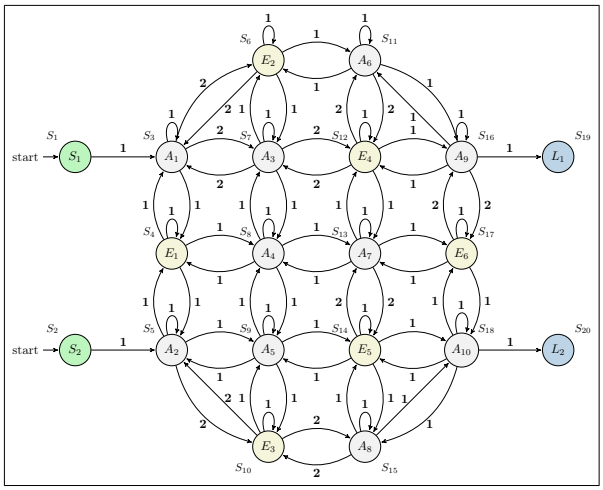

Fig. 4: The Inspection environment

To further demonstrate the efficiency of the proposed verification algorithm, we consider case study II which resembles a real-world end-to-end robotic solution that automates industrial inspections [2]. In this case study, robots are used to monitor complex installations of energy and industrial processing plants to provide up-to-date and reliable data on plant machinery to enhance industrial operations. Plant operators use the collected data to maximize equipment uptime, enhance operations, and improve safety while reducing operations costs. Consider the floor of the plant to be routinely inspected in Figure 4 which is composed of 2 initial positions S_1 and S_2, 6 equipment to be routinely inspected $E_1, \cdots, E_6$, 2 landing stations L_1 and L_2, and 10 allowable states $A_1, \cdots, A_{10}$. On each inspection routine, the robot starts from any of the initial states (green), proceeds to collect data from installed equipment (yellow) on the plant floor, and then finally to the landing states (blue). We abstract the inspection environment into a weighted graph where the nodes represent the initial states, landing stations, allowable states, and installed equipment, while the edges represent transitions between the nodes, and the assigned weights represent travel times associated with the transitions. Based on this case study, we consider 3 different scenarios with 3 different HyperTWTL specifications that include optimality, symmetry, and linearizability.

Requirement 5 (Shortest path): Optimality requirements are important hyperproperties in robotic applications. One such requirement is finding the shortest path over the human-robot collaboration environment. Let us consider a scenario where a robot starts the inspection from any of the initial states S_1 or S_2, followed by an inspection and gathering data from equipment E_1, E_2, E_3, E_4, E_5, and E_6. After gathering data from all installed equipment, the robot finally proceeds to any of the landing states L_1 and L_2. Given a set of executions, there exists an execution mapped to π_2 that reaches a landing state from the initial states before any other execution mapped to π_1. This can be formalized as a synchronous HyperTWTL formula φ_5 as shown in Table 8.

Table 9: Verification results of HyperTWTL properties for Case Study II

HyperTWTL Req.	Description	Verdict	Z3		CVC4		AMC	
			Time (Seconds)	Memory (MB)	Time (Seconds)	Memory (MB)	Time (Seconds)	Memory (MB)
φ_5	Shortest Path	SAT	5.112	10.423	4.587	12.401	18.143	19.860
φ_6	Symmetry	SAT	5.985	10.258	4.937	12.537	18.353	20.315
φ_7	Linearizability	UNSAT	5.674	10.695	4.185	12.118	19.058	21.865

Requirement 6 (Symmetry): Let us assume that two robots are available to inspect and gather data from equipment $E_1, \cdots E_6$. In this case, one robot must inspect and gather data from equipment E_1, E_2, E_4, and E_6, while the other robot should inspect and gather data from equipment E_1, E_3, E_5, and E_6. We assume E_2 and E_4 are mapped to π_1 if and only if E_3 and E_5 are already mapped to π_2 and vice-versa. This can be formalized as a synchronous HyperTWTL formula φ_6 as shown in Table 8.

Requirement 7 (Linearizability): The principle underlying linearizability is that the whole system operates as if executions from all human-robot collaborations are from one collaboration. Thus, linearizability is a correctness condition that guarantees consistency across concurrent executions of a given system. 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 essential to ensure that the mission's primary goal to inspect and gather data from installed equipment is completed before proceeding to the landing states L_1 or L_2 is not violated. This can be formalized as a synchronous HyperTWTL formula φ_7 as in Table 8.

4.4 Case Study II: Experimental Results

All experiments are performed in the same computing environment and follow the same procedure as case study I. The following time bounds are considered for the verification of all the HyperTWTL properties in Table 8: $T_1 = 1$, $T_2 = 2$, $T_3 = 4$, $T_4 = 5$, $T_5 = 8$, $T_6 = 9$, $T_7 = 13$, $T_8 = 14$, $T_9 = 19$, $T_{10} = 20$, $T_{11} = 23$, $T_{12} = 24$, $T_{13} = 26$, $T_{14} = 27$, and $T_{15} = 29$. Since the time bounds are the same for HyperTWTL formulae from φ_5–φ_7, their unrolling bound is also the same, i.e., $||\varphi|| = 129$. Similar to case study 1, we verify the HyperTWTL specification using Z3 and CVC4 SMT solvers and compare them with the automata-based model checking (AMC) approach in [9]. The obtained results for case study II are shown in Table 9. From Table 9, we observe that φ_5 and φ_6 were satisfied using Z3, CVC4 and AMC whereas φ_7 was unsatisfied. Once again, we observe that the verification time never exceeded 5.985 seconds in Z3, 4.937 seconds in CVC4, and 19.058 seconds in AMC. This shows a 3.55× and 3.96× speed up for Z3 and CVC4, respectively, regarding execution time. We also observe that while verifying the above specifications, the memory consumed never exceeded 10.695MB in Z3, 12.537 MB in CVC4, and 21.865 MB in AMC.

Table 10: Scalability Analysis for SMT-based Model Checking for HyperTWTL

TKS size	Unrolling bound $\|\|\varphi\|\|$	Z3		CVC4		AMC	
		Time (s)	Memory (MB)	Time (s)	Memory (MB)	Time (s)	Memory (MB)
20^2	51	1.45	8.60	0.84	10.33	16.10	17.93
20^4	100	115.46	25.81	73.35	42.10	692.20	151.62
20^6	150	1128.04	105.83	619.07	219.76	2145.82	802.73
20^8	200	4500.88	296.32	2721.56	532.91	-	-
20^{10}	250	11978.13	1074.41	8859.02	1933.84	-	-

Once again, this shows that our SMT-based verification approach is 2× and 1.8× more memory-efficient while using Z3 and CVC4, respectively.

5 Scalability Analysis

In our last set of experiments, we evaluate the scalability of our proposed verification approach by varying the size of TKS $\mathcal{T}$ and verifying them using our approach vs. the AMC approach. The size of the $\mathcal{T}$ ranges from 20^2 to 20^{10} with randomly generated transitions. For this experiment, we consider φ_2 as the specification and vary the unrolling bound $||\phi||$ in the range of 50–250. All experiments are performed in the same computing environment as case studies I and II. The obtained results for the scalability analysis are presented in Table 10. Table 10 shows that for φ_2, the verification time increases with the increasing size of the TKS. However, the results shown in Table 10 also suggest that our proposed verification approach using SMT solvers, i.e., Z3 and CVC4, are more scalable than the previously proposed AMC approach. For instance, for the TKS with 20^2 states and $||\phi|| = 50$, it takes only 1.45 and 0.84 seconds for Z3 and CVC4, respectively for verification. However, verifying the same φ_2 using AMC takes 16.10. This shows approximately 11× and 19× speedup for our approach compared to the AMC. Similarly, for a TKS with 20^6 states and $||\phi|| = 150$, Z3 and CVC4 takes 1128.04 seconds and 619.07 seconds, respectively to verify φ_2. Verifying the same requirement using the AMC approach requires 2145.82 seconds. This shows approximately 2× and 3.5× speedup for our approach compared to the AMC. The comparison of memory consumption for verification also follows the same trend. Interestingly, while verifying the TKSs with 20^8 and 20^{10} states for $||\phi|| = 200$ and $||\phi|| = 250$, the AMC approach experienced a state-space explosion. In contrast, our proposed SMT approach successfully verified the property using Z3 and CVC4 in 4500.88 and 2721.56 seconds, respectively.

Bounded model checking with SMT has been successfully used in developing safety-critical industrial systems for decades [4, 29]. We believe that engineers can use our proposed HyperTWTL model checking approach to verify a wide range of safety and security properties of large-scale, complex, and safety-critical robotic missions.

6 Related Works

Model checking [11] has extensively been used to verify hyperproperties of models abstracted as transition systems by examining their related state transition graphs [10]. In [15], the first model checking algorithms for HyperLTL and HyperCTL* employing alternating automata were proposed, which was also adopted in [8, 21] to verify HyperMTL properties. An extensive study on the complexity of verifying hyperproperties with model checking is presented in [5]. The bounded model checking approach has recently become popular in the verification of HyperLTL specifications [16, 22, 23, 27]. Specifically, the work in [31] is most relevant to ours, where the authors use HyperLTL and an SMT solver for robotic mission planning. The work in [31] was indeed the first attempt to use hyperproperties for robotic mission planning. However, HyperLTL cannot express tasks with explicit time constraints, which motivates our contribution in this paper. Very recently, the authors in [9] proposed the HyperTWTL formalism and an automata-based model checking approach for verifying them. In contrast to [9], this paper presents a bounded model checking approach for verifying HyperTWTL specification using SMT solvers for enhanced verification performance regarding verification time and memory.

7 Conclusion

This paper introduced a bounded model checking approach for HyperTWTL using SMT solvers, contrasting the existing automata-based HyperTWTL verification. Specifically, we reduce the HyperTWTL model checking problem to a first-order logic satisfiability problem and then use two state-of-the-art SMT solvers, i.e., Z3 and CVC4, for verification. Using two case studies, Technical Surveillance Squadron (TESS) and Robotic Industrial Inspection, and a scalability study, we showed that the proposed bounded model checking approach can efficiently verify HyperTWTL properties compared to the AMC approach while offering up to 19× speed up in terms of verification time and up to 2× memory efficiency. Our scalability analysis results also show that our proposed approach can verify large systems, whereas the previously reported automata-based HyperTWTL verification method suffers from a state-space explosion.

References

1. Alpern, B., et. al: Defining liveness. Information processing letters **21**(4), 181–185 (1985)
2. ANYbotics: Automation Digitalization at Scale: ANYmal Makes the Case at BASF. https://www.anybotics.com/anymal-makes-the-case-at-basf-chemical-plant/ (2021)
3. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking (2003)
4. Bjørner, N.: Z3 and smt in industrial r&d. In: FM. pp. 675–678. Springer (2018)

5. Bonakdarpour, B., et. al: The complexity of monitoring hyperproperties. In: 2018 IEEE 31st CSF. pp. 162–174. IEEE (2018)
6. Bonakdarpour, B., et. al: Monitoring hyperproperties by combining static analysis and runtime verification. In: ISoLA. pp. 8–27. Springer (2018)
7. Bonakdarpour, B., et. al: Controller synthesis for hyperproperties. In: 2020 IEEE 33rd CSF. pp. 366–379. IEEE (2020)
8. Bonakdarpour, B., et. al: Model checking timed hyperproperties in discrete-time systems. In: NASA Formal Methods Symposium. pp. 311–328. Springer (2020)
9. Bonnah, E., et. al: Model checking time window temporal logic for hyperproperties. In: 21st ACM-IEEE MEMOCODE. pp. 100–110 (2023)
10. Clarke, E., et. al: Bounded model checking using satisfiability solving. "FM" **19**(1), 7–34 (2001)
11. Clarke, E.M.: Model checking. In: FSTTCS. pp. 54–56. Springer (1997)
12. Clarkson, M.R., et. al: Temporal logics for hyperproperties. In: POST. pp. 265–284. Springer (2014)
13. Clarkson, M.R., Schneider, F.B.: Hyperproperties. Journal of Computer Security **18**(6), 1157–1210 (2010)
14. Coenen, N., et. al: Verifying hyperliveness. In: CAV. pp. 121–139. Springer (2019)
15. Finkbeiner, B., et. al: Algorithms for model checking hyperltl and hyperctl. In: CAV. pp. 30–48. Springer (2015)
16. Finkbeiner, B., et. al: Specifying and verifying secrecy in workflows with arbitrarily many agents. In: ATVA. pp. 157–173. Springer (2016)
17. Finkbeiner, B., et. al: Model checking quantitative hyperproperties. In: CAV. pp. 144–163. Springer (2018)
18. Finkbeiner, B.e.a.: Eahyper: satisfiability, implication, and equivalence checking of hyperproperties. In: CAV. pp. 564–570. Springer (2017)
19. Garey, M.R., et. al: Computers and intractability. A Guide to the (1979)
20. Goguen, J.A., et. al: Security policies and security models. In: 1982 IEEE Symposium on Security and Privacy. pp. 11–11. IEEE (1982)
21. Ho, H.M., et. al: On verifying timed hyperproperties. arXiv preprint arXiv:1812.10005 (2018)
22. Hsu, T.H., et. al: Hyperqube: A qbf-based bounded model checker for hyperproperties. arXiv preprint arXiv:2109.12989 (2021)
23. Hsu, T.H., Sánchez, C., Bonakdarpour, B.: Bounded model checking for hyperproperties. In: TACAS. pp. 94–112. Springer (2021)
24. Knightscope: Knightscope Deploys New Autonomous Security Robot in Southern California. https://www.businesswire.com/news/home/20220316005436/en/Knightscope-Deploys-New-Autonomous-Security-Robot-in-Southern-California (2022)
25. Nguyen, L.V., et. al: Hyperproperties of real-valued signals. In: MEMOCODE. pp. 104–113 (2017)
26. Paviot-Adet, E., et. al: Structural reductions and stutter sensitive properties. arXiv preprint arXiv:2212.04218 (2022)
27. Pommellet, A., et. al: Model-checking hyperltl for pushdown systems. In: SPIN. pp. 133–152. Springer (2018)
28. Romano, S.A.: Persistent surveillance gives squadron its global purpose, https://www.af.mil/News/Article-Display/Article/1152329/persistent-surveillance-gives-squadron-its-global-purpose/
29. Rungta, N.: A billion smt queries a day. In: CAV. pp. 3–18. Springer (2022)
30. Vasile, C.I., et. al: Time window temporal logic. Theoretical Computer Science **691**, 27–54 (2017)

31. Wang, Y., et. al: Hyperproperties for robotics: Planning via hyperltl. In: 2020 IEEE ICRA. pp. 8462–8468. IEEE (2020)
32. Zdancewic, S., et. al: Observational determinism for concurrent program security. In: 16th IEEE CSF. pp. 29–43. IEEE (2003)

Appendix

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.

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* [26]. 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 $\Sigma_\epsilon = \Sigma \cup \epsilon$. 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 $inv(\mathbb{T})$.

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 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; $\beta_{[\varphi]^{X,Y}} = up(X) + up(Y)$ 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.