Multi-Agent Path Planning with Hyperproperties

Anonymous Author(s)

ABSTRACT

In this paper, we introduce a logic-based discrete path planning technique for *multi-agent* robotic systems. The novelty of our approach stems from the theory of hyperproperties. A *hyperproperty* is a set of sets of execution traces, expressing the property of a set of executions, rather then traditional trace-based languages that describe properties of individual executions. The inherent multi-trace nature of hyperproperties makes them a highly desirable framework to specify objectives and intended behaviors of controllable robots and uncontrollable adversaries in multi-agent systems. We specify several path planning problems in the temporal logic HyperLTL— a generalization of the Linear Temporal Logic (LTL) that allows for simultaneous and explicit quantification over execution traces. We show that HyperLTL can capture many important path planning requirements such as optimization, privacy, priority, and adversarial control policies. We reduce the path planning problem to the HyperLTL verification problem for bounded executions and utilize a solution based on solving the satisfiability problem for quantified Boolean formulas (QBF). Our experiments show the applicability of our method in terms of generality and the diversity of path planning problems as well as efficiency and effectiveness of our approach.

KEYWORDS

Formal methods for CPS, Hyperproperties, Multi-robot planning, Robotics, Temporal Logic

ACM Reference Format:
Anonymous Author(s). 2020. Multi-Agent Path Planning with Hyperproperties. In *Proceedings of Woodstock '18: ACM Symposium on Neural Gaze Detection (ICCPS '21).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Formal methods are a class of automated and rigorous techniques that have important applications in the design and implementation of cyber-physical systems (CPS). These systems often carry out complex tasks, such as functions of robots and autonomous vehicles. Specifically, the complex tasks of a system are formally expressed in some form of temporal logic (e.g., the linear temporal logic (LTL) [22] or signal temporal logic (STL) [18]), which provide a set of well-defined syntactic and semantic rules. Based on this, automatic *correct-by-construction* algorithms can be designed to compute control policies for general tasks expressed by temporal logics — traditionally by model-based synthesis techniques [3] and more recently using model-free learning [5, 11**?** , 12].

An increasing number of CPS are built in a *multi-agent* or distributed fashion [10, 15, 26, 30]. In these systems, multiple agents implement *local* control policies with partial or complete information on the whole system, in order to collaboratively carry out a *global* task. Historically, most global tasks that have been studied are relatively simple, such as consensus [23] and formation [19].

Recently, there is a growing interest in using formal methods to design control policies for complex logical tasks of multi-agent CPS. To formally capture these tasks by temporal logics, common temporal logics, such as LTL and STL, meet a stumbling block: they can only express tasks for a single execution of a single agent. Therefore, to use them for multi-agent CPS, in the most ideal case, one has to explicitly decompose the global task into several relatively independent local tasks, and use LTL or STL to capture each local task [15, 17, 21]. For tasks that cannot be decomposed into local tasks, counting LTL (cLTL) [16, 24, 25] and TeamLTL [27] are proposed to augment the syntactic and semantic rules of common temporal logic to reason about global tasks that simultaneously involve the executions from multiple agents. However, all the above works do not support important global tasks that involve *explicit* quantification to individual trajectories. For example, they cannot specify a global task such as "for any execution of agent A, there exists a control policy for Agent B to fulfill a joint temporal logic task with Agent A".

To capture global task requirements in multi-agent applications, and following a recent study in [28], we propose a general and novel technique based on *hyperproperties* [7] as our tool to specify objectives of path planning. Technically speaking, a hyperproperty is a set of sets of execution traces that specify a system-wide property, rather than the property of individual traces. The theory of hyperproperties was first introduced as a framework to deal with information-flow security policies that need to reason about multiple executions simultaneously [**?**]. Since then, temporal logics HyperLTL [6] and HyperPCTL [1, 2] have been introduced to provide clear formal syntax and semantics to different classes of hyperproperties. Besides information-flow control, hyperproperties have been a powerful tool to express and reason about consistency in concurrent data structures [4] as well as sensitivity and robustness in CPS [29].

In this paper, we employ HyperLTL as an elegant and expressive tool to seamlessly express a rich set of requirements of global and local objectives in multi-agent robotic applications and subsequently solve the corresponding path planning problems for a given system. HyperLTL extends LTL by allowing explicit and simultaneous quantification over a set of execution paths that have some relation with each other. For example, consider the problem of finding the shortest path in a single-agent setting. The objective can be expressed as the HyperLTL formula:

$$\exists\pi.\forall\tau.(\neg goal_\tau \ \mathcal{U} \ goal_\pi).$$

ICCPS '21, June 03–05, 2018, Woodstock, NY
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The formula essentially captures that path π (i.e., the shortest path) reaches the goal before any other path τ.

In the multi-agent setting, we focus on two classes of path planning problems in (1) *adversarial*, and (2) *non-adversarial* applications. In the non-adversarial scenarios, the agents fulfill the assigned tasks with no interference from other agents. For example, in the *squad shift* problem, a squad of n robots need to take turns to constantly occupy a location (in an arbitrary order). The HyperLTL formulas for non-adversarial problems are usually of the form:

$$\exists\pi_1.\exists\pi_2\cdots\exists\pi_n.\psi,$$

where path variables $\pi_1,\ldots,\pi_n$ are instantiated to satisfy the objective ψ.

On the other hand, in the adversarial setting, a set of *controllable* agents should plan their paths knowing that another set of *uncontrollable* agents may unintentionally or maliciously interfere with their control actions. For example, an adversarial robot can act as a *moving obstacle*, attempting to block the paths of the controllable robots to achieve their goal. The HyperLTL formula for adversarial problems are normally of the form:

$$\forall\pi'_1.\forall\pi'_2\cdots\forall\pi'_m.\exists\pi_1.\exists\pi_2\cdots\exists\pi_n.\psi,$$

where path variables $\pi_1\cdots\pi_n$ are instantiated to satisfy the objective ψ in the presence of *any* adversarial paths $\pi'_1\cdots\pi'_m$.

The path planning algorithm introduced in this work takes as input (1) a HyperLTL formula φ that specifies the global control objectives of the agents, and (2) a discrete-time transition system (DTS) D that models the agents' operation. The algorithm synthesizes as output a set of paths as the witness to the existential quantifiers that satisfy φ. Specifically, our synthesis algorithm works as follows. First, we adopt the bounded terminating semantics for HyperLTL proposed in [14]. It allows us to reason about terminating systems such as path planning. The bounded terminating semantics also allow us to reduce our synthesis problem to the satisfiability problem for *quantified Boolean formulas* (QBF). In other words, the QBF encoding $[\![D,\varphi]\!]$ is satisfiable if and only if the answer to the path planning problem is affirmative.

We have fully implemented the introduced technique and conducted a rich set of experiments. Our non-adversarial path planning problems include *squad shift* (i.e., constant occupation of a critical region by at least one robot at each time instant), *priority* (i.e., some robot has priority over another to be in a region), as well as *robustness* (i.e., independence from the start region) and *opacity* (i.e., privacy-preserving exploration) from [28]. In the adversarial setting, we showed applicability our method on *fairness* (i.e., a robot is not discriminated by another), *moving obstacle* (i.e., a robot actively attempts to block another to reach its goal), and *the pursuer-evader* game from [8], where the pursuer's goal is to catch the evader in a sensor network. Our experiments show the effectiveness of our approach as well as its efficiency in spite of its generality.

Organization. In Section 2, we introduce the preliminary concepts. The formal statement of the path planning problem and the corresponding synthesis algorithm is introduced in Section 3. We discuss the set of path planning problems in Section 4, and evaluate our experimental results for these problems in Section 5. Related work is discussed in Section 6. Finally, we make concluding remarks and discuss avenues for future work in Section 7.

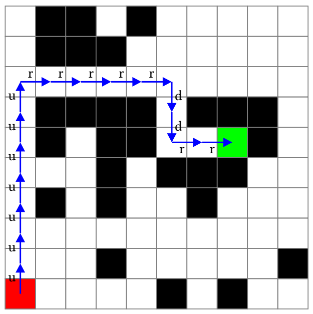

Figure 1: Shortest Path in a 10×10 grid.

2 PRELIMINARIES

2.1 Discrete Transition Systems

We model the system where the robots operate as discrete transition systems. Let AP be a finite set of *atomic propositions* and $\Sigma = 2^{\mathsf{AP}}$ be the *alphabet*. A *letter* is an element of Σ. A *trace* $t\in\Sigma^\omega$ over alphabet Σ is an infinite sequence of letters: $t = t(0)t(1)t(2)\cdots$.

Definition 2.1. A *discrete transition system* (DTS) is a tuple $D=\langle S, S_{init}, \mathsf{Act}, \delta, \mathsf{AP}, L\rangle$, where

- S is a finite set of *states*;
- $S_{init}\subseteq S$ is the set of *initial states*;
- Act is a finite set of *actions*;
- $\delta\subseteq S\times\mathsf{Act}\times S$ is a *transition relation*;
- AP is the set of atomic propositions, and
- $L: S\longrightarrow\Sigma$ is a *labeling function* on the states of D.

We require that for each $s\in S$, there exists $s'\in S$, such that $(s,a,s')\in\delta$, for some $a\in\mathsf{Act}$. We also assume that δ is deterministic with respect to the choice of actions.

For example, the 10×10 grid shown in Figure 1 can be modeled by a DTS with 100 states, where the red and green cells are the initial and goal states, respectively. A robot can change state by a set of actions $\mathsf{Act}=\{r,l,u,d\}$ denoting right, left, up, and down moves on the grid. The black cells are obstacles, preventing a robot to reach them. Thus, the actions in Act are not enabled in all states.

Let us denote a transition (s,a,s') by $s\xrightarrow{a}s'$. A *loop* in D is a finite sequence $s(0)s(1)\cdots s(n)$, such that $s(i)\xrightarrow{a_i}s(i+1)\in\delta$, for all $0\le i<n$, and $s(n)\xrightarrow{a_n}s(0)\in\delta$. We call a DTS *acyclic*, if the only loops are self-loops on otherwise *terminal* states, i.e., on states that have no other outgoing transition. In this paper, we only consider acyclic DTS, where terminal states are labeled by a special atomic proposition *halt*.

A *path* of a DTS is an infinite sequence of states of the form:

$$s(0)\xrightarrow{a_0}s(1)\xrightarrow{a_1}\cdots,$$

such that $s(0)\in S_{init}$, and $s(i)\xrightarrow{a_i}s(i+1)\in\delta$, for all $i\ge0$. A trace of a DTS is a trace $t(0)t(1)t(2)\cdots\in\Sigma^\omega$, such that there exists a path $s(0)\xrightarrow{a_0}s(1)\xrightarrow{a_1}\cdots$ with $t(i)=L(s(i))$, for all $i\ge0$. We denote by *Traces*(D) the set of all traces of DTS D. For example, the blue arrows in Figure 1 illustrate a path in the DTS that starts from the initial state and consists of seven up actions, followed by five right, two down, and another two right actions, and finally reaches the goal state.

2.2 The Temporal Logic HyperLTL

HyperLTL [6] is an extension of the linear-time temporal logic (LTL) for hyperproperties. In this paper, since our path planning problem is for finite and terminating paths, we utilize the *bounded terminating semantics* introduced in [14] to deal with terminating protocols.

2.2.1 *Syntax.*

The syntax of HyperLTL formulas is defined inductively by the following grammar:

$$\varphi ::= \exists\pi.\varphi \mid \forall\pi.\varphi \mid \phi$$
$$\phi ::= \mathsf{true} \mid p_\pi \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi\,\mathcal{U}\,\phi \mid \phi\,\mathcal{R}\,\phi \mid \bigcirc\phi$$

where $p \in \mathsf{AP}$ is an atomic proposition and π is a *trace variable* from an infinite supply of variables $\mathcal{V}$. The Boolean connectives $\neg$, $\vee$, and $\wedge$ have the usual meaning, $\mathcal{U}$ is the temporal *until* operator, $\mathcal{R}$ is the temporal *release* operator, and $\bigcirc$ is the temporal *next* operator. We also consider other derived Boolean connectives, such as $\rightarrow$, and $\leftrightarrow$, and the derived temporal operators *eventually* $\Diamond\varphi \equiv \mathsf{true}\ \mathcal{U}\ \varphi$ and *globally* $\Box\varphi \equiv \neg\Diamond\neg\varphi$. The quantified formulas $\exists\pi$ and $\forall\pi$ are read as "along some trace π" and "along all traces π", respectively.

A formula is *closed* (i.e., a *sentence*) if all trace variables used in the formula are quantified. We assumed, without lost of generality that no variable is quantified twice. We use $\mathit{Vars}(\varphi)$ for the set of trace variables used in formula φ.

2.2.2 *Semantics.*

Interpretation. An interpretation $\mathcal{T} = \langle T_\pi \rangle_{\pi \in \mathit{Vars}(\varphi)}$ of a formula φ consists of a set of traces, one set T_π per trace variable π in $\mathit{Vars}(\varphi)$. We use T_π for the set of traces assigned to π. The idea here is to allow quantifiers to range over different systems. We use this feature to handle path planning for robots that have related but different controllable constrains or plants (e.g., adversarial robots and in particular, our pursuer-evader example). Another reason is that we may need different agents to be able to adopt independent control policies.

With this feature, each set of traces comes from its own DTS and we use $\mathcal{D} = \langle D_\pi \rangle_{\pi \in \mathit{Vars}(\varphi)}$ to denote a *family* of discrete transition systems. Thus, $T_\pi = \mathit{Traces}(D_\pi)$ is one set of traces that π can range over, which comes from D_π. Abusing notation, we write $\mathcal{T} = \mathit{Traces}(\mathcal{D})$. Note that all trace sets being the same set of traces for a single DTS D is a particular case that all robots have the same controllable constraints (i.e. robust path synthesis), which leads to the original HyperLTL. The multi-model nature of our interpretation allows us to incorporate heterogeneous robots that can potentially operate in different plants.

Terminating Semantics. To adapt to robotic path planning with given time horizons, we use a terminating semantics for HyperLTL. This semantics is the *halting pessimistic* introduced in [14] for generating bounded model checking queries to verify HyperLTL formulas. The semantics are defined with respect to a trace assignment, which is a partial map $\Pi\colon \mathit{Vars}(\varphi) \rightharpoonup \Sigma^\omega$. The assignment with an empty domain is denoted by $\Pi_\emptyset$. In the sequel, let us denote the set of integers $\{1, 2, \ldots, n\}$ with interval $[1, n]$. We assume the HyperLTL formula is closed and of the form:

$$\mathbb{Q}_1\pi_1.\mathbb{Q}_2\pi_2 \ldots \mathbb{Q}_n\pi_n.\psi$$

where each $\mathbb{Q}_i \in \{\forall, \exists\}$, for $i \in [1, n]$, and it has been converted into negation-normal form (NNF) so that the negation symbol only appears in front of atomic propositions, e.g., $\neg p_{\pi_1}$. Let $\mathcal{T} = \langle T_1 \ldots T_n \rangle$ be a tuple of finite sets of finite traces, one per trace variable. Given a trace assignment Π, a trace variable π, and a concrete trace $t \in \Sigma^\omega$, we denote by $\Pi[\pi \rightarrow t]$ the assignment that coincides with Π everywhere but at π, which is mapped to trace t.

We start by defining a satisfaction relation between HyperLTL formulas within a finite discrete *time horizon* h (i.e., the number of steps in a trace which is obviously bounded by the longest path in a DTS), and models $(\mathcal{T}, \Pi, i)$, where $\mathcal{T}$ is the tuple of set of traces, Π is a trace assignment mapping, and $i \in \mathbb{Z}_{\geq 0}$ that points to each position of traces. The satisfaction of a HyperLTL formula φ is a binary relation $\models_h$ that associates a formula to the models $(\mathcal{T}, \Pi, i)$ where $i \in \mathbb{Z}_{\geq 0}$ is a pointer that indicates the current position of all traces in $\mathcal{T}$.

We define the terminating semantics separately in quantifiers, Boolean operators, and temporal operators, as follows:

Quantifiers. The satisfaction relation for the quantifiers is:

$$\begin{array}{lll}
(\mathcal{T}, \Pi, 0) \models_h \exists\pi.\ \psi & \text{iff} & \text{there is a } t \in T_\pi \\
 & & \quad \text{such that } (\mathcal{T}, \Pi[\pi \rightarrow t], 0) \models_h \psi \\
(\mathcal{T}, \Pi, 0) \models_h \forall\pi.\ \psi & \text{iff} & \text{for all} \quad t \in T_\pi \\
 & & \quad \text{such that } (\mathcal{T}, \Pi[\pi \rightarrow t], 0) \models_h \psi
\end{array}$$

Boolean operators. For every $i \leq h$, we have that:

$$\begin{array}{lll}
(\mathcal{T}, \Pi, i) \models_h \mathsf{true} & & \\
(\mathcal{T}, \Pi, i) \models_h p_\pi & \text{iff} & p \in \Pi(\pi)(i) \\
(\mathcal{T}, \Pi, i) \models_h \neg p_\pi & \text{iff} & p \notin \Pi(\pi)(i) \\
(\mathcal{T}, \Pi, i) \models_h \psi_1 \vee \psi_2 & \text{iff} & (\mathcal{T}, \Pi, i) \models_h \psi_1 \text{ or } (\mathcal{T}, \Pi, i) \models_h \psi_2 \\
(\mathcal{T}, \Pi, i) \models_h \psi_1 \wedge \psi_2 & \text{iff} & (\mathcal{T}, \Pi, i) \models_h \psi_1 \text{ and } (\mathcal{T}, \Pi, i) \models_h \psi_2
\end{array}$$

Temporal connectives. If $(i < h)$, we follow the normal temporal operator in HyperLTL semantics:

$$\begin{array}{lll}
(\mathcal{T}, \Pi, i) \models_h \bigcirc\psi & \text{iff} & (\mathcal{T}, \Pi, i+1) \models_h \psi \\
(\mathcal{T}, \Pi, i) \models_h \psi_1\,\mathcal{U}\,\psi_2 & \text{iff} & (\mathcal{T}, \Pi, i) \models_h \psi_2, \text{ or} \\
 & & (\mathcal{T}, \Pi, i) \models_h \psi_1 \text{ and} \\
 & & (\mathcal{T}, \Pi, i+1) \models_h \psi_1\,\mathcal{U}\,\psi_2 \\
(\mathcal{T}, \Pi, i) \models_h \psi_1\,\mathcal{R}\,\psi_2 & \text{iff} & (\mathcal{T}, \Pi, i) \models_h \psi_2, \text{ and} \\
 & & (\mathcal{T}, \Pi, i) \models_h \psi_1 \text{ or} \\
 & & (\mathcal{T}, \Pi, i+1) \models_h \psi_1\,\mathcal{R}\,\psi_2
\end{array}$$

For $(i = h)$, we define the following for deciding eventualities. To capture the terminating semantics, we use the predicate *halt* that is true if the state that corresponds to a terminating state (self-loop), and define $\mathit{Halt} \overset{\text{def}}{=} \bigwedge_{\pi \mathit{Vars}(\varphi)} \mathit{halt}_\pi$ which holds whenever all traces have terminated (and their final state will be repeated ad infinitum). Thus, we have:

$$\begin{array}{lll}
(\mathcal{T}, \Pi, i) \models_h \bigcirc\psi & \text{iff} & (\mathcal{T}, \Pi, i) \models_h \mathit{Halt} \text{ and} \\
 & & (\mathcal{T}, \Pi, i) \models_h \psi, \\
(\mathcal{T}, \Pi, i) \models_h \psi_1\,\mathcal{U}\,\psi_2 & \text{iff} & (\mathcal{T}, \Pi, i) \models_h \psi_2 \\
(\mathcal{T}, \Pi, i) \models_h \psi_1\,\mathcal{R}\,\psi_2 & \text{iff} & (\mathcal{T}, \Pi, i) \models_h \psi_1 \wedge \psi_2, \text{ or} \\
 & & (\mathcal{T}, \Pi, i) \models_h \mathit{Halt} \text{ and } (\mathcal{T}, \Pi, i) \models_h \psi_2
\end{array}$$

We say that an interpretation $\mathcal{T}$ satisfies a specification φ, denoted by $\mathcal{T} \models_h \varphi$, if $(\mathcal{T}, \Pi_\emptyset, 0) \models_h \varphi$. We say that a family of DTSs $\mathcal{D}$ satisfies a specification φ, denoted by $\mathcal{D} \models_h \varphi$, if it holds that $\langle \mathit{Traces}(D_\pi) \rangle_{\pi \in \mathit{Vars}(\varphi)} \models_h \varphi$.

For example, the *shortest path* as shown in Figure 1 is a planning objective for one agent which can be expressed in HyperLTL as:

$$\varphi_{\mathrm{sp}} = \exists \pi. \forall \tau. (\neg \mathit{goal}_\tau \ \mathcal{U} \ \mathit{goal}_\pi).$$

That is, all paths τ reach the goal at or after the point that the shortest path π reaches the goal. It is straightforward to see that in the DTS in Figure 1, the path denoted by the blue arrows is the shortest path from the initial state to the goal and, hence, a satisfying witness to π. Our goal in this paper is to reduce the path planning problem to the HyperLTL verification problem, where the witness to existential trace quantifiers establish paths for different agents that satisfy a common objective.

3 PROBLEM STATEMENTS AND QBF-BASED SOLUTION

We now formally state the path planning problem through a reduction to the HyperLTL verification problem.

3.1 Formal Statement of the Problem

We start with the following definitions.

Definition 3.1. Let $D = \langle S, S_{init}, \mathrm{Act}, \delta, \mathrm{AP}, L \rangle$ be a DTS. We say that $A = a_0 a_1 \cdots$ is a *valid* sequence of actions if there exists a path $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \cdots$ in D.

Definition 3.2. Let $D = \langle S, S_{init}, \mathrm{Act}, \delta, \mathrm{AP}, L \rangle$ be a DTS. A *control policy* is a function $c : 2^{\mathrm{Act}^*} \longrightarrow 2^{\mathrm{Act}^*}$ that maps a set of valid finite sequences of actions to another set of valid sequences of actions.

The intuition behind Definition 3.2 is that a control policy maps a finite set of actions (typically by adversaries) to another finite set of sequences of actions (typically by controllable agents) that collectively satisfy a global objective.

We assume *n controllable* agents and *m uncontrollable* adversaries. Let $\mathcal{D}$ be a family of DTSs and $\psi(\pi_1 \cdots \pi_n, \pi'_1 \cdots \pi'_m)$ be a HyperLTL formula describing the objective constraints over the behavior of the agents and adversaries, where $\pi_1 \cdots \pi_n$ and $\pi'_1 \cdots \pi'_m$ are free trace variables in ψ. Clearly, in a non-adversarial setting, the paths $\pi'_1 \cdots \pi'_m$ are omitted. For instance, in the aforementioned shortest path example we have:

$$\psi_{\mathrm{sp}} = \forall \tau. (\neg \mathit{goal}_\tau \ \mathcal{U} \ \mathit{goal}_\pi),$$

where π is the free trace variable (to be computed during path planning).

Intuitively, the path planning problem is to find a control function c that maps any set of sequences of actions of adversaries to some set of sequences of actions of the agents that satisfy a global control objective ψ. This problem is analogous to solving the following HyperLTL verification problem.

Given a family of DTS $\mathcal{D}$, m adversaries, n agents, and a global control objective formula ψ, does there exist a control policy such that:

$$\mathcal{D} \models \underbrace{\forall \pi'_1. \forall \pi'_2 \cdots \forall \pi'_m}_{\text{adversaries}} . \underbrace{\exists \pi_1. \exists \pi_2 \cdots \exists \pi_n}_{\text{agents}} . \underbrace{\ \psi\ }_{\text{objective}} \ ;$$

here, each π'_i is the valid path for a sequence of actions $a'^i_0 a'^i_1 a'^i_2 \cdots$ of adversary $i \in [1, m]$, and each π_j is the valid path for a sequence of actions $a^j_0 a^j_1 a^j_2 \cdots$ of agent $j \in [1, n]$.

For example, for the shortest path objective with no adversaries, the path planning problem is mapped into solving the verification problem with respect to the formula:

$$\varphi_{\mathrm{sp}} = \exists \pi. \underbrace{\forall \tau. (\neg \mathit{goal}_\tau \ \mathcal{U} \ \mathit{goal}_\pi)}_{\psi_{\mathrm{sp}}} .$$

In this case, π provides us with the sequence of actions that a robot should take to follow the shortest path. Clearly, when there is no adversary, paths $\pi_1, \pi_2, \ldots, \pi_n$ are computed such that they satisfy ψ.

3.2 Reduction to the QBF Satisfiability Problem

We solve the multi-agent planning problem using a reduction to the *quantified Boolean satisfiability* (QBF) [9] checking problem. That is, given a family of DTS $\mathcal{D}$, a HyperLTL planning objective φ, and a time horizon h, we construct a QBF formula $[\![\mathcal{D}, \varphi]\!]_h$ that is satisfiable if and only the answer to the path planning problem is affirmative. The QBF satisfiability problem is the following:

> *Given is a set of Boolean variables, $X = \{x_1, x_2, \ldots, x_n\}$, and a formula $F = \mathbb{Q}_1 x_1. \mathbb{Q}_2 x_2 \ldots \mathbb{Q}_{n-1} x_{n-1}. \mathbb{Q}_n x_n. \psi$, where each $\mathbb{Q}_i \in \{\forall, \exists\}$ ($i \in [1, n]$) and ψ is an arbitrary Boolean formula over variables X. Is F true?*

Figure 2 shows a satisfying model for the given QBF.

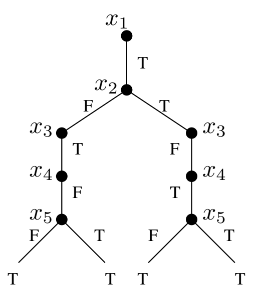

Figure 2: Model for the QBF formula $F = \exists x_1. \forall x_2. \exists x_3. \exists x_4. \forall x_5. (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_3 \vee x_4 \vee \neg x_5) \wedge (x_1 \vee x_4 \vee x_5)$

Our approach is an adaptation of the HyperLTL bounded model checking algorithm proposed in [14]. More specifically, let φ be a HyperLTL formula of the form $\varphi = \mathbb{Q}_1 \pi_1. \mathbb{Q}_2 \pi_2. \ldots . \mathbb{Q}_n \pi_n. \psi$ and $\mathcal{D} = \langle D_1, D_2, \ldots, D_n \rangle$. The encoding of the HyperLTL verification problem in QBF is the following:

$$[\![\mathcal{D}, \varphi]\!]_h = \mathbb{Q}_1 \overline{d_1}.\mathbb{Q}_2 \overline{d_2} \cdots .\mathbb{Q}_n \overline{d_n} \Big([\![D_1]\!]_h \circ_1 [\![D_2]\!]_h \circ_2 \cdots [\![D_n]\!]_h \circ_n [\![\psi]\!]_h \Big),$$

where $[\![\psi]\!]_h$ is the encoding of the inner LTL formula ψ, $[\![D_j]\!]$ is the encoding of DTS D_j, $\circ_j = \wedge$ if $\mathbb{Q}_j = \exists$ and $\circ_j = \rightarrow$ if $\mathbb{Q}_j = \forall$, for $j \in [1, n]$. The encodings of $[\![D_j]\!]_h$ and $[\![\psi]\!]$ as propositional formulas are identical to that of standard bounded model checking. Since in this paper, our focus is on finite path planning, time horizon h is guaranteed to be bounded. In particular, when the unrolling of D_j reaches a state labeled by *halt*, the unrolling can stop. While the details of the transformation is outside the scope of this paper, we explain the construction of $[\![\mathcal{D}, \varphi]\!]_h$ using an example.

3.3 Example

3.3.1 Encoding of DTS. Consider the grid in Figure 1 modeled by a DTS D. The size of the set d of variables needed for Boolean representation of D is $\log_2(10) + \log_2(10) + 2$, to encode x and y coordinates and two propositions *goal* and *halt*. Thus, the initial state s_0 with coordinate (0, 0) is encoded as:

$$I(s_0) := (\neg x_3 \wedge \neg x_2 \wedge \neg x_1 \wedge \neg x_0) \wedge (\neg y_3 \wedge \neg y_2 \wedge \neg y_1 \wedge \neg y_0) \wedge \neg goal \wedge \neg halt. \quad (1)$$

All other states are similarly encoded. The transition relation of the DTS is denoted by $R(d^i, d^{i+1})$ for each unrolling step $i \leq h$, where d^i is a fresh copy of Boolean variables in d. Therefore, the transition relation up to bound h, is the following:

$$[\![D]\!]_h := I(d^0) \wedge R(d^0, d^1) \wedge R(d^1, d^2) \wedge \cdots \wedge R(d^{h-1}, d^h).$$

where I is the initial state condition defined in equation (1). Finally, for each trace variable $\pi \in Vars(\varphi)$, we introduce a fresh copy of $\{d^0, d^1 \cdots, d^h\}$, which we denote by $\overline{d_\pi}$.

3.3.2 Encoding of Inner LTL Formula. To encode the inner LTL formula, we use the fixpoint representation of the formula in the same fashion as bounded model checking. For example, for the shortest path formula, we have:

$$\begin{aligned}[\![\neg goal_\tau \,\mathcal{U}\, goal_\pi]\!]_h := goal^0_\pi \vee (\neg goal^0_\tau \wedge \\ (goal^1_\pi \vee (\neg goal^1_\tau \wedge \\ (goal^2_\pi \vee (\neg goal^2_\tau \wedge \\ \cdots \\ (goal^{h-1}_\pi \vee goal^h_\pi))))))\end{aligned}$$

3.3.3 Complete Formula. Finally, we are ready to construct the complete QBF query for the DTS, formula φ_{sp}, and time horizon h; this is achieved as follows:

$$[\![\mathcal{D}, \varphi_{\mathsf{sp}}]\!]_h := \exists \overline{d_\pi}.\forall \overline{d_\tau}.\Big([\![D_\pi]\!] \wedge ([\![D_\tau]\!] \rightarrow [\![\neg goal_\tau \,\mathcal{U}\, goal_\pi]\!]_h)\Big).$$

Now, for the DTS in Figure 1, checking the satisfiability of final formula $[\![\mathcal{D}, \varphi_{\mathsf{sp}}]\!]_h$ results in returning the truth assignments to variables in $\overline{d_\pi}$ as witness of satisfiability. Indeed, this assignment represents the path of blue arrows as π, which is the shortest path.

4 MULTI-AGENT PATH PLANNING OBJECTIVES IN HYPERLTL

In this section, we present various multi-robot planning problems whose objectives can be formally expressed by HyperLTL via the formulation introduced in Section 3. These objectives cannot be expressed by either cLTL [16] or TeamLTL [27]. We organize this section by first presenting the non-adversarial cases in Section 4.1 and subsequently the path planning problems in the adversarial setting in Section 4.2.

4.1 Non-Adversarial Planning

A common class of multi-robot planning problems is to check whether a (global) objective is achieved for all possible paths of the robots. Formally, these problems can be expressed by a HyperLTL formula of the form $\exists \pi_1 \cdots \exists \pi_n.\psi$. The global objective ψ itself can be a HyperLTL formula of the form $\psi = \mathbb{Q}\tau_1 \cdots \mathbb{Q}\tau_k.\ \varphi$ that contains quantifications over auxiliary paths $\tau_1 \cdots \tau_k$. As the problem statement in Section 3.1 requires, our goal is to identify an assignment for path variables $\pi_1, \ldots, \pi_n$ to satisfy ψ.

4.1.1 ***Squad Shift.*** Consider N robots working in a shared space (e.g., for surveillance). Suppose we need the rotation of a squad of n robots ($n \leq N$) to constantly occupy a location *crit* (e.g., a critical region) in arbitrary orders. Such a multi-robot planning problem can be captured by:

$$\begin{aligned}\varphi_{\mathsf{sqs}} &= \exists \pi_1 \cdots \exists \pi_n.\ \psi_{\mathsf{sqs}}, \quad (2) \\ \psi_{\mathsf{sqs}} &= \Box \bigwedge_{i=1}^{n} \Big(crit_{\pi_i} \rightarrow \bigcirc \bigvee_{j=1, j\neq i}^{n} crit_{\pi_j} \Big),\end{aligned}$$

where the objective ψ is an unquantified HyperLTL and our goal is to synthesize the paths $\pi_1 \cdots \pi_n$ for a squad of n robots to achieve the objective. For simplicity, we omit the task of the other $N - n$ robots in (2). The subformula ψ_{sqs} means if robot i is currently in the region *crit*, then there exists another robot j ($j \neq i$) from the squad to replace the robot i next. Observe that formula (2) cannot be captured by cLTL [16, 24] when $n < N$. This is because although cLTL can reason about the *number* of robots in the region *crit*, it cannot explicitly specify the behavior of a robot from a squad of n robots. This is, of course, possible in HyperLTL, since the logic allows explicit path quantification.

4.1.2 ***Priority.*** Consider n robots working in a shared space (e.g., in a warehouse). Suppose they need to pass a region B (e.g., a one-way bridge) with two ends BL and BR. To avoid congestion, if two robots are to pass B from two opposite directions (i.e. two robots passing B from BL and BR, respectively) at the same time, then the robot that arrives first should pass the region B first. This multi-robot planning problem can be captured by

$$\begin{aligned}\varphi_{\mathsf{pri}} &= \exists \pi_1 \cdots \exists \pi_n.\ \psi_{\mathsf{pri}} \quad (3) \\ \psi_{\mathsf{pri}} &= \bigwedge_{i=1}^{n} \Big(\Diamond B_{\pi_i} \wedge \bigwedge_{j=1, i\neq j}^{n} \Box \big(\rho(\pi_i, \pi_j) \wedge \rho(\pi_j, \pi_i)\big) \Big) \\ \rho(\pi_i, \pi_j) &= (BL_{\pi_i} \wedge \Diamond BR_{\pi_j}) \rightarrow (\neg B_{\pi_j} \,\mathcal{U}\, B_{\pi_i}),\end{aligned}$$

where the subformula $\rho(\tau_i, \pi_j)$ means if a robot first arrives at $BL\,/BR$ (to pass B), then another robot arrives later at the other end (i.e. $BR\,/BL$) should wait until the first-arriving one passes.

4.1.3 ***Opacity [28].*** Consider a team of n robots working to fulfilling a global task $\rho(\pi_1, \ldots, \pi_n)$. Opacity aims at keeping sensitive information on the paths of the robots secure while fulfilling the task. In other words, given two sets of paths $\pi_1 \cdots \pi_n$ and $\tau_1 \cdots \tau_n$, where each set satisfies a global objective ρ and each pair π_i and τ_i (for all $i \in [1, n]$) that start from different initial states, opacity requires that each step of the pairs reach equivalent states, as far as AP is concerned. Formally:

$$\begin{aligned}\varphi_{\text{iso}} &= \exists\pi_1 \cdots \exists\pi_n.\ \psi_{\text{iso}} \\ \psi_{\text{iso}} &= \exists\tau_1 \cdots \exists\tau_n.\ \bigwedge_{i=1}^{n} \neg(\pi_i[0] \leftrightarrow \tau_i[0]) \ \wedge \\ &\qquad \rho(\pi_1, \ldots, \pi_n) \ \wedge\ \rho(\tau_1, \ldots, \tau_n) \ \wedge \\ &\qquad \bigwedge_{i=1}^{n} \bigwedge_{p \in \mathsf{AP}} \Box(p_{\pi_i} \leftrightarrow p_{\tau_i})\end{aligned} \tag{4}$$

More specifically, the first conjunct requires that the initial states of π_i and τ_i are different, the second and third conjuncts require $\pi_1 \cdots \pi_n$ and $\tau_1 \cdots \tau_n$ to satisfy ρ, and the last conjunct requires that each pair π_i and τ_i reach indistinguishable states. We note that the last conjunct can technically replace AP with actions in Act (by abuse of notation) if the objective of opacity is to have equivalent sequences of actions instead of states. When (4) holds, there are (at least) two sets of paths from different initial states to achieve the task ρ, so the exact initial states of the paths $\pi_1, \ldots, \pi_n$ to fulfill $\rho(\pi_1, \ldots, \pi_n)$ is opaque. Formula 4 φ_{iso} generalizes the notion of opacity for single robot planning in [28].

Likewise, the opacity of the robots' paths can be expressed by a formula where the initial states of pairs of π_i and τ_i are equivalent, but they follow different paths:

$$\begin{aligned}\varphi_{\text{po}} &= \exists\pi_1 \cdots \exists\pi_n.\ \psi_{\text{po}} \\ \psi_{\text{po}} &= \exists\tau_1 \cdots \exists\tau_n.\ \bigwedge_{i=1}^{n} (\pi_i[0] \leftrightarrow \tau_i[0]) \ \wedge \\ &\qquad \rho(\pi_1, \ldots, \pi_n) \ \wedge\ \rho(\tau_1, \ldots, \tau_n) \ \wedge \\ &\qquad \neg \bigwedge_{i=1}^{n} \bigwedge_{p \in \mathsf{AP}} \Box(p_{\pi_i} \leftrightarrow p_{\tau_i})\end{aligned} \tag{5}$$

When (5) holds, there are (at least) two sets of paths from the same initial states to achieve the task ρ, so the exact the paths $\pi_1, \ldots, \pi_n$ taken to fulfill $\rho(\pi_1, \ldots, \pi_n)$ is opaque.

4.1.4 ***Robustness [28].*** When controlling a team of robots, the knowledge of their initial states are subject to bounded (sensing) errors, i.e., the real initial states are in a neighborhood of the known states. The objective of fulfilling a global task $\rho(\pi_1, \ldots, \pi_n)$ (involving n robots) under bounded errors in the initial states can be expressed in HyperLTL by the following formula:

$$\begin{aligned}\varphi_{\text{rob}} &= \exists\pi_1 \cdots \exists\pi_n.\psi_{\text{rob}}, \\ \psi_{\text{rob}} &= \forall\tau_1 \cdots \forall\tau_n.\ \bigwedge_{i=1}^{n} \mathit{neighbor}_{\pi_i} \ \wedge\ \mathit{neighbor}_{\tau_i} \ \wedge \\ &\qquad \rho(\pi_1, \ldots, \pi_n) \ \wedge\ \rho(\tau_1, \ldots, \tau_n) \ \wedge \\ &\qquad \bigwedge_{i=1}^{n} \bigwedge_{p \in \mathsf{AP}} \Box(p_{\pi_i} \leftrightarrow p_{\tau_i}).\end{aligned} \tag{6}$$

where *neighbor* is a proposition indicating a particular neighborhood. Thus, the first conjunct means that initial states of π_i and τ_i are located in the same neighborhood. The meaning of the second and third conjuncts are the same as (4). When (6) holds, the task $\rho(\pi_1, \ldots, \pi_n)$ can be robustly fulfilled under perturbations on the initial states of $\pi_1, \ldots, \pi_n$. Formula 6 φ_{rob} generalizes the notion of robustness for single robot planning in [28].

4.2 Adversarial Path Planning

More generally than Section 4.1, we consider the planning problem of n robots to a (global) objective in the presence of m uncontrollable adversarial robots. As mentioned in Section 3.1, these problems can be expressed by a HyperLTL formula of the form:

$$\forall\pi'_1 \cdots \forall\pi'_m.\ \exists\pi_1 \cdots \exists\pi_n.\ \psi,$$

where the objective is of the form $\psi = \mathbb{Q}\tau_1 \ldots \mathbb{Q}\tau_k.\ \varphi$ that contains quantifications over auxiliary paths $\tau_1, \ldots, \tau_k$. Our goal is to identify a control policy that decides the assignment of the path variables $\pi_1, \ldots, \pi_n$ to satisfy ψ from any possible assignment of the adversarial path variables $\pi'_1 \ldots \pi'_m$.

4.2.1 ***Moving obstacle.*** Consider a team of $n + m$ robots, where n robots working to fulfilling a task $\rho(\pi_1, \ldots, \pi_n)$. To ensure the task ρ is carried out without collisions, regardless of the paths $\pi'_1, \ldots, \pi'_m$ of the other m robots (which can be viewed as moving obstacles), the control policy should satisfy the following HyperLTL formula:

$$\begin{aligned}\varphi_{\text{obs}} &= \forall\pi'_1 \cdots \forall\pi'_m.\exists\pi_1 \cdots \exists\pi_n.\ \psi_{\text{obs}} \\ \psi_{\text{obs}} &= \rho(\pi_1, \ldots, \pi_n) \ \wedge\ \bigwedge_{i=1}^{n+m} \bigwedge_{j=1, i\neq j}^{n+m} \bigwedge_{p \in \mathsf{AP}} \Box\neg(p_{\pi_i} \leftrightarrow p_{\tau_i}),\end{aligned} \tag{7}$$

where the conjunct indicates the robots do not collide. When (7) holds, the m robots can achieve the task ρ without collisions.

4.2.2 ***Fairness.*** Consider several robots working in a shared space (e.g., in a warehouse). Suppose B is a region with limited access that requires no matter what path one robot chooses (the adversary π'), another robot, whose path is π can eventually follow the adversary to access the shared area. This multi-robot planning problem can be captured by the following formula in HyperLTL:

$$\begin{aligned}\varphi_{\text{fair}} &= \forall\pi'.\exists\pi.\ \psi_{\text{fair}} \\ \psi_{\text{fair}} &= \Box\left(B_{\pi'} \wedge \bigcirc\neg B_{\pi'} \rightarrow \bigcirc(\neg B_{\pi'}\ \mathcal{U}\ B_{\pi})\right)\end{aligned} \tag{8}$$

where the objective ψ_{fair} means that path π' should not enter the region twice before π enters.

4.2.3 ***Pursuer-Evader Game [8].***

Given a sensor network consisting of several motes, where each mote is able to carry certain information, such as clock time or if some agent is currently detected on this mote. Also, each mote can interact with its neighbors, which allows to build a parent-child relationship and form a *tracking tree*. Now, given two agents an *evader* and a *pursuer*, we assume that the evader can move freely to escape from the pursuer with the ability to get full knowledge of every mote on the entire sensor network (e.g., the current location of the pursuer). On the other hand, the pursuer is limited to only be able to obtain the information from the mote that it currently resides on. Our goal is to synthesize (1) a *network design*, such that each mote can detect the presence of agents and transmitting certain information to its neighbors, and (2) the behavior of the pursuer to eventually catch the evader. The specification of the game in HyperLTL is as follows:

$$\begin{aligned}\varphi_{pe} = \forall\pi_1.\exists\pi_2.\exists\pi_3.\ &\Box(pos_{\pi_1} \leftrightarrow pos\_evader_{\pi_2}) \wedge \\ &\Box(mote\_info_{\pi_2} \leftrightarrow mote\_info_{\pi_3}) \wedge \\ &\Diamond(pos_{\pi_1} \leftrightarrow pos_{\pi_3})\end{aligned} \tag{9}$$

where *pos* is the position of the agent, *pos_evader* is the position of the mote that detects the evader on the network, *mote_info* encodes the information that each mote can contain/transmit, including time stamp and parent information. We note that the universal quantification on $\forall\pi_1$ allows the evader to choose any path, the state evolution of the sensor network is identified by $\exists\pi_2$, and the behavior of the is pursuer synthesized through $\exists\pi_3$.

5 EVALUATION

5.1 Experimental Settings

We have implemented the technique described in Section 3.2. In this section, we describe this implementation and the empirical evaluation of the case studies described in Section 4. Our implementation works as follows. Given a DTS, we automatically unfold it up to a given time horizon $h \geq 0$ using a home-grown tool written in Ocaml. The unfolded DTS is combined with the QBF encoding of the input HyperLTL formula to form a complete QBF instance which is then fed to the QBF solver Quabs [13]. All experiments in this section are run on a MacBook Pro laptop with Intel i7 CPU @2.8 GHz and 16 GB of RAM.

We experiment with different path planning objectives, number of agents, and size of DTS. In the following sections, we use colors red for initial state(s), green for goal state(s), black for obstacles, yellow for bridge area, also green for areas on two ends of the bridge *BL* and *BR* .

5.2 Non-Adversarial Case Studies

Recall in the non-adversarial case, the HyperLTL formula starts with a sequence of existential quantifiers. A satisfiability result from the QBF solver implies successful synthesis, where the collection of witness traces to the existential quantifiers form a valid global control policy that satisfies the global control objective. The results are summarized in Table 1. In a nutshell, in cases where the formula involves only existential quantifiers, we observe a high degree of scalability. However, for path robustness, since the formula involves quantifier alternation, the synthesis time grows quickly.

5.2.1 ***Squad Shift Planning.***

Figure 3 shows the result of synthesizing paths when the number of agents is two, i.e., $N = n = 2$ in formula (2), for a 10×10 grid, where the critical region is labeled by proposition *crit* in the purple cell. Witness paths for π_1 and π_2 are represented by blue and orange arrows, respectively. As can be seen, each agent eventually reaches the critical section and π_1 and π_2 paths take turn to guard *crit*. The results for different sizes of grids and agents are shown in Table 1. We note that for the case of $N = n = 2$, the squad shift objective can be written as $\Box\Diamond[BL, 1]$ in cLTL [25]. But generally when $N \neq n$, the planning objective (2) cannot be expressed by cLTL, since they cannot distinguish the identities of the agents guarding the area.

5.2.2 ***Priority Planning.***

Figure 4 shows the result of synthesizing paths when the number of agents in formula (3) is two for a 10×10 grid. The synthesized paths π_1 and π_2 start at the initial red cell and are shown in blue and orange arrows, respectively. Since π_1 reaches *BL* before π_2 , it is given the priority to pass the bridge before π_2 .

Next, in Fig 5, we demonstrate another scenario when π_1 is required to complete the emergency mission at the pink area before passing the bridge. This is enforced in the HyperLTL formula. In this case, because π_1 still reached *BL* before π_2 , we can observe that π_2 moves around and waits on the left side of the bridge, and does not passing bridge until π_1 did, in order to preserve the priority specification. The results for different sizes of grids and agents are shown in Table 1.

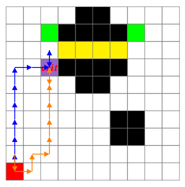

Figure 3: Paths satisfying φ_{sqs}.

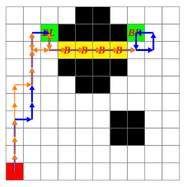

Figure 4: Agents satisfying φ_{pri}.

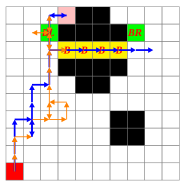

Figure 5: Agents satisfying φ_{pri}.

5.2.3 ***Opacity Planning.***

Here we investigate two different concepts of opacity in robotic planning.

Initial-State opacity (ISO). Assume now we give agents the mission of reaching the goal states (colored in green) in Fig. 6. Recall from formula (4) that the ISO property requires that each agent π_i has another corresponding τ_i, such that starting from a different initial state, both can reach the goal by following the same sequence of actions. For two agents, we substitute ρ by $(\Diamond\, goal_{\pi_1} \wedge \Diamond\, goal_{\pi_2})$ and $(\Diamond\, goal_{\tau_1} \wedge \Diamond\, goal_{\tau_2})$ for π and τ paths in formula (4). The synthesis result is presented in Fig. 6 for two agents, where π_1 and τ_1 paths red and pink, and π_2 and τ_2 paths are blue and green. Clearly, we cannot identify agents by observing their initial state behavior.

Path opacity (PO). Following the same mission setting in ISO and replacement of ρ in formula (5), now we consider having the paths in the shaded area as sensitive information in Fig. 7. Thus, the concrete formula for PO with two agents is the following:

$$\begin{aligned}\varphi_{\text{po}} = \quad & \exists\pi_1.\exists\pi_2.\exists\tau_1.\exists\tau_2.\\ & (\Diamond goal_{\pi_1} \wedge \Diamond goal_{\pi_2}) \wedge (\Diamond goal_{\tau_1} \wedge \Diamond goal_{\tau_2}) \wedge \\ & \neg\Box(Act_{\pi_1} = Act_{\tau_1}) \wedge \neg\Box(Act_{\pi_2} = Act_{\tau_2}) \wedge \\ & \Box((shaded_{\pi_1} \wedge shaded_{\tau_1}) \rightarrow (Act_{\pi_1} = Act_{\tau_1})) \wedge \\ & \Box((shaded_{\pi_2} \wedge shaded_{\tau_2}) \rightarrow (Act_{\pi_2} = Act_{\tau_2}))\end{aligned}$$

As shown in 7, π_1 and τ_1 in red and pink are two different paths, but their actions are the same throughout the *shaded* area. Similarity, paths π_2 and τ_2 satisfy this specification as well. The results for different sizes of grids and agents are shown in Table 1.

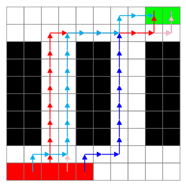
Figure 6: φ_{iso} of initial states for multi-agent

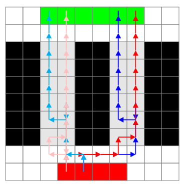
Figure 7: φ_{po} of path multi-agent

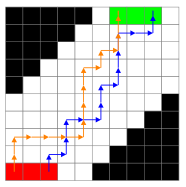
Figure 8: φ_{rob} for multi-robots

5.2.4 *Robustness Planning*

Our goal is to synthesize two agents, π_1 and π_2, such that the actions for each agent, are robust in terms of all other traces on the map. Similar to opacity, we replace ρ in formula (6) with $(\Diamond goal_{\pi_1} \wedge \Diamond goal_{\pi_2})$ and $(\Diamond goal_{\tau_1} \wedge \Diamond goal_{\tau_2})$ for π and τ paths. As shown in Figure 8, the paths synthesized for the agents represent a valid robust strategy for all other paths on the map. The results for different sizes of grids and agents are shown in Table 1. As mentioned earlier, due to the nature of the HyperLTL formula, the synthesis time grows significantly faster than other path planning problems.

5.3 Adversarial Case Studies

We now focus on path planning problems involving adversaries. The results are summarized in Table 2. Recall that HyperLTL formulas for adversarial cases are alternating with a leading universal quantifier (i.e., $\forall^+\exists^+$). When our synthesis technique returns satisfiability, it does not enumerate a witness to existential quantifiers that follow universal quantifiers, due to a combinatorial explosion. However, for sanity check, our tool can spit out a witness for a particular instantiation of the universal quantifiers.

5.3.1 *Moving Obstacles Planning*

Given the grid shown is Figs. 9 and 10, there are adversaries for which no successful control policy exists (e.g., in black arrows in Fig. 10) and there are ones for which there is a control policy (e.g., in black arrows in Fig. 9). This means in general, for such a grid, the problem is not satisfiable for all adversaries. However, the problem has a solution for the grid in Fig. 11. Our technique can identify all such cases.

5.3.2 *Fairness Planning.*

We experiment with the grid shown in Fig. 12 with respect to formula (8) φ_{obs}. Our algorithm returns a satisfiability result, meaning that for all adversarial paths π_1, there is another path π_2 that satisfies the formula.

Prop	# agents	*QS*	*size*	*h*	Total[s]
φ_{sqs}	2	$\exists^2$	10^2	15	**1.04**
			20^2	25	**6.78**
			40^2	35	**107.43**
	4	$\exists^4$	10^2	20	**4.85**
			20^2	30	**30.49**
			40^2	45	**170.52**
	8	$\exists^8$	10^2	25	**25.85**
			20^2	40	**94.82**
			40^2	55	**474.15**
	10	$\exists^{10}$	10^2	30	**39.01**
			20^2	50	**175.93**
			40^2	65	**864.46**
φ_{pri}	2	$\exists^2$	10^2	20	**2.74**
			20^2	40	**22.11**
			40^2	70	**178.10**
	4	$\exists^4$	10^2	25	**14.04**
			20^2	50	**66.80**
			40^2	90	**601.11**
	8	$\exists^8$	10^2	35	**63.29**
			20^2	60	**220.67**
			40^2	110	**1872.45**
	10	$\exists^{10}$	10^2	45	**97.30**
			20^2	75	**450.00**
			40^2	140	**4446.63**
φ_{rob}	1	$\exists\forall$	10^2	17	**0.46**
			20^2	38	**28.23**
			40^2	80	**1208.34**
	2	$\exists^2\forall$	10^2	18	**0.89**
			20^2	38	**33.07**
			40^2	80	**1394.49**
φ_{iso}	1	$\exists^2$	10^2	17	**0.55**
			20^2	38	**21.67**
			40^2	80	**457.09**
	2	$\exists^4$	10^2	18	**1.30**
			20^2	38	**39.57**
			40^2	80	**917.24**
	4	$\exists^8$	20^2	38	**103.32**
	6	$\exists^{12}$			**165.31**
	10	$\exists^{20}$			**267.64**
φ_{po}	1	$\exists^2$	10^2	13	**0.31**
			20^2	38	**24.13**
			40^2	40	**470.36**
	2	$\exists^4$	10^2	15	**0.77**
			20^2	38	**44.46**
			40^2	80	**1028.55**
	4	$\exists^8$	20^2	38	**103.46**
	6	$\exists^{12}$			**158.77**
	10	$\exists^{20}$			**250.14**

Table 1: Results for non-adversarial experiments. $\exists^k$ denotes k number of existential quantifiers in a row.

One particular example is shown in Fig. 12, where the blue adversary passes the bridge the first time and when it attempts to

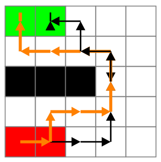
Figure 9: **Agent beats adversary for φ_{obs}.**

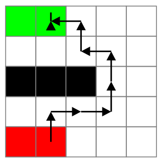
Figure 10: An unbeatable adversary for φ_{obs}.

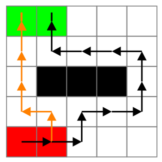
Figure 11: Agent beats adversary for φ_{obs}.

Prop	QS	$size$	h	Total[s]
φ_{obs}	$\forall\exists$	10^2	13	**1.31**
		20^2	35	**45.03**
		40^2	55	**840.68**
φ_{fair}	$\forall\exists$	10^2	30	**3.63**
		20^2	60	**44.82**
		40^2	100	**129.69**
φ_{pegame}	$\forall\exists\exists$	3^2	7	**0.49**

Table 2: Results for adversarial experiments.

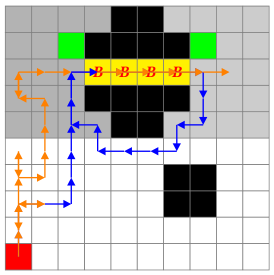

Figure 12: π_1 passes the bridge twice with φ_{fair} preserved.

pass again, the orange controllable agent forces the adversary to wait until it passes the bridge.

5.3.3 ***Pursuer-Evader Game.*** In our implementation, we use three different DTSs to represent the behaviors of the evader, sensor network, and pursuer. The pursuer-evader game is a perfect example of the application of our multi-model framework introduced in Section 2 for path planning.

As mentioned in Section 4.2, the evader is free to choose any trajectory (i.e., $\forall\pi_1$ in formula (9)). One scenario that we synthesized is the following. In Fig. 13, initially, the evader is in cell 1 and the pursuer is in cell 9. Next, the evader moves to cells 4 and 5 (see Fig. 14). This move will result in updating the network design to form a tracking tree by using the timestamp of evader's entrance in these cells and assigning the parent-child relationship for $\exists\pi_2$ in formula (9). That is, at $h = 2$, we have $parent(2) = 1$, $parent(1) = parent(7) = 4$, and $parent(4) = 5$ (the tracking tree is rooted at the location of the evader). In Fig. 15 and 16, the evader moves to cells 2 and then 3. Since we assume that the pursuer moves faster than the evader, the tracking tree in Fig. 16 leads the pursuer (i.e., by $\exists\pi_3$ in formula (9)) through cells 9, 8, 5, 2, and 3 to catch the evader.

Our synthesis technique returns a satisfiability result, meaning that for all evader moves, we can compute a correct tracking tree for the pursuer to catch the evader. Indeed, our algorithm generates the same "evader centric" solution proposed in [8].

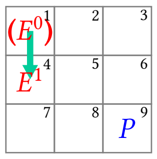

Figure 13: $h = 1$

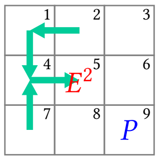

Figure 14: $h = 2$

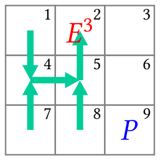

Figure 15: $h = 3$

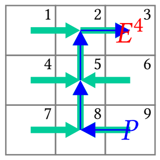

Figure 16: $h = 4$

5.4 Summary of Results

Tables 1 and 2 summarize our experimental results. Two important observations from these tables are the following. First, although the underlying PSPACE-complete problem of QBF-solving can be a potential stumbling block, our experiments show the effectiveness of our approach for moderate-sized models and number of agents. Secondly, as expected, the path planning problem is generally more difficult to solve for cases involving quantifier alternation. Even in that case, we are not limited to small models.

We also note that we have replicated the majority of the experiments in [25] using our approach. For reasons of space, we are not able to present a detailed comparison and contrast. However, it is noteworthy that as expected, the cLTL approach slightly outperforms our algorithm, since HyperLTL is more expressive, making our technique more general, but slightly slower.

6 RELATED WORK

Planning with non-hyper temporal logics. Most previous work on formal methods for multi-agent cyber-physical or robotic systems focuses on non-hyper temporal logics, such as LTL and STL [15, 17, 21]. However, due to the drawback that these logics can only express tasks for a single execution of a single agent, these works depend on explicitly decomposing the global task into several relatively independent local tasks so that they can use LTL or STL to capture each local task. To express non-decomposable global tasks, cLTL [16, 24, 25] extends LTL by adding quantification to specify the number of states satisfying certain sub-formulas along paths. cLTL allows simultaneous reasoning over executions from multiple agents and can capture tasks such as "there are always at least two agents in the same region". For cLTL, effective synthesis algorithms can be derived by SMT solving. TeamLTL [27] extends LTL by allowing atomic propositions that describe the properties of several robots at a time instance. This work is the extension of cLTL and TeamLTL to allow explicit existential and universal quantifications over paths, to handle tasks like "for any execution of agent A, there exists a control policy for agent B to fulfill a joint temporal logic task with agent A".

Path planning with hyper temporal logic. To allow explicit existential and universal quantifications over paths, one needs a hyper temporal logic such as HyperLTL. In [28], HyperLTL is first applied to the planning of a single robot for objectives that implicitly involves multiple paths, such as opacity (to plan a path that yields another obfuscating path) and robustness (to plan a path such that is robust to perturbations). Different from [28], this paper applies

HyperLTL to *multi-robot* planning and studies richer classes of planning objectives, especially planning with adversaries (as discussed in Section 4.2). To adapt to planning with finite time horizons, we used a terminating semantics with an explicitly specified time horizon for HyperLTL, as opposed to finite-trace semantics in [28?]. Furthermore, this work proposes a new QBF-based design method for HyperLTL objectives, which is practically more efficient than the SMT-based design methods proposed in [28].

7 CONCLUSION

In this paper, we proposed a novel technique for multi-agent path planning for robotic applications by using hyperproperties expressed in HyperLTL. We showed that HyperLTL can elegantly express important path planning objectives in both adversarial and non-adversarial settings. Our path planning algorithm is based on a bounded model checking technique for HyperLTL proposed in [14]. Our algorithm reduces the path planning problem to the satisfiability problem for quantified Boolean formulas. Through a set of experiments, we showed (1) the diversity of path planning problems that our technique can handle, and (2) the effectiveness and efficiency of approach in spite of its generality.

As for future work, there are several interesting open research problems. Our current approach lacks finding cyclic paths. This is mainly because enforcing a loop condition in bounded model checking for hyperproperties is not trivial at all. Another important problem is to generalize our hyperproperty-based technique to enable path planning in continuous signal settings (e.g., by using the temporal logic HyperSTL [20]).

REFERENCES

[1] E. Ábrahám, E. Bartocci, B. Bonakdarpour, and O. Dobe. 2020. Probabilistic Hyperproperties with Nondeterminism. In *Proceedings of the 18th Symposium on Automated Technology for Verification and Analysis (ATVA)*. 518–534.

[2] E. Ábrahám and B. Bonakdarpour. 2018. HyperPCTL: A Temporal Logic for Probabilistic Hyperproperties. In *Proceedings of the 15th International Conference on Quantitative Evaluation of Systems (QEST)*. 20–35.

[3] Rajeev Alur. 2015. *Principles of Cyber-Physical Systems*. The MIT Press, Cambridge, Massachusetts.

[4] B. Bonakdarpour, C. Sánchez, and G. Schneider. 2018. Monitoring Hyperproperties by Combining Static Analysis and Runtime Verification. In *Proceedings of the 8th Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*. 8–27.

[5] Alper Kamil Bozkurt, Yu Wang, Michael Zavlanos, and Miroslav Pajic. 2020. Control Synthesis from Linear Temporal Logic Specifications Using Model-Free Reinforcement Learning. In *IEEE International Conference on Robotics and Automation (ICRA)*. Paris, France, 10349–10355.

[6] M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez. 2014. Temporal Logics for Hyperproperties. In *Proceedings of the 3rd Conference on Principles of Security and Trust POST*. 265–284.

[7] M. R. Clarkson and F. B. Schneider. 2010. Hyperproperties. *Journal of Computer Security* 18, 6 (2010), 1157–1210.

[8] M. Demirbas, A. Arora, and M. G. Gouda. 2003. A Pursuer-Evader Game for Sensor Networks. In *Proceedings of the 6th International Symposium on Self-Stabilizing Systems (SSS)*. 1–16.

[9] M.R. Garey and D.S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York.

[10] Ritwika Ghosh, Joao P. Jansch-Porto, Chiao Hsieh, Amelia Gosse, Minghao Jiang, Hebron Taylor, Peter Du, Sayan Mitra, and Geir Dullerud. 2020. CyPhyHouse: A Programming, Simulation, and Deployment Toolchain for Heterogeneous Distributed Coordination. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 6654–6660.

[11] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. 2019. Omega-Regular Objectives in Model-Free Reinforcement Learning. In *Tools and Algorithms for the Construction and Analysis of Systems*, Tomáš Vojnar and Lijun Zhang (Eds.). Vol. 11427. Springer International Publishing, Cham, 395–412.

[12] Mohammadhosein Hasanbeig, Yiannis Kantaros, Alessandro Abate, Daniel Kroening, George J. Pappas, and Insup Lee. 2019. Reinforcement Learning for Temporal Logic Control Synthesis with Probabilistic Satisfaction Guarantees. In *IEEE 58th Conference on Decision and Control (CDC)*. 5338–5343. arXiv:1909.05304

[13] J. Hecking-Harbusch and L. Tentrup. 2018. Solving QBF by Abstraction. In *Proceedings of the 9th International Symposium on Games, Automata, Logics, and Formal Verification (GandALF) (EPTCS)*, Vol. 277. 88–102.

[14] Tzu-Han Hsu, César Sánchez, and Borzoo Bonakdarpour. 2020. Bounded Model Checking for Hyperproperties. *CoRR* abs/2009.08907 (2020). arXiv:2009.08907 http://arxiv.org/abs/2009.08907

[15] Yiannis Kantaros, Meng Guo, and Michael M. Zavlanos. 2019. Temporal Logic Task Planning and Intermittent Connectivity Control of Mobile Robot Networks. *IEEE Trans. Automat. Control* 64, 10 (Oct. 2019), 4105–4120.

[16] Francois Laroussinie, Antoine Meyer, and Eudes Petonnet. 2010. Counting LTL. In *2010 17th International Symposium on Temporal Representation and Reasoning*. 51–58.

[17] Lars Lindemann, Jakub Nowak, Lukas Schönbächler, Meng Guo, Jana Tumova, and Dimos V. Dimarogonas. 2019. Coupled Multi-Robot Systems Under Linear Temporal Logic and Signal Temporal Logic Tasks. *IEEE Transactions on Control Systems Technology* (2019), 1–8.

[18] Oded Maler and Dejan Nickovic. 2004. Monitoring Temporal Properties of Continuous Signals. In *FTRTFT*. 152–166.

[19] Nathan Michael, Michael M. Zavlanos, Vijay Kumar, and George J. Pappas. 2008. Distributed Multi-Robot Task Assignment and Formation Control. In *2008 IEEE International Conference on Robotics and Automation*. 128–133.

[20] L. V. Nguyen, J. Kapinski, X. Jin, J. V. Deshmukh, and T. 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 (MEMOCODE)*. 104–113.

[21] Y. V. Pant, H. Abbas, R. A. Quaye, and R. Mangharam. 2018. Fly-by-Logic: Control of Multi-Drone Fleets with Temporal Logic Objectives. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. 186–197.

[22] Amir Pnueli. 1977. The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science (Sfcs 1977)*. 46–57.

[23] Wei Ren and R.W. Beard. 2005. Consensus Seeking in Multiagent Systems under Dynamically Changing Interaction Topologies. *IEEE Trans. Automat. Control* 50, 5 (May 2005), 655–661.

[24] Yunus Emre Sahin, Petter Nilsson, and Necmiye Ozay. 2017. Synchronous and Asynchronous Multi-Agent Coordination with cLTL+ Constraints. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. 335–342.

[25] Yunus Emre Sahin, Petter Nilsson, and Necmiye Ozay. 2020. Multirobot Coordination With Counting Temporal Logics. *IEEE Transactions on Robotics* 36, 4 (Aug. 2020), 1189–1206.

[26] Hussein Sibai, Navid Mokhlesi, Chuchu Fan, and Sayan Mitra. 2020. Multi-Agent Safety Verification Using Symmetry Transformations. In *Tools and Algorithms for the Construction and Analysis of Systems (Lecture Notes in Computer Science)*, Armin Biere and David Parker (Eds.). Springer International Publishing, Cham, 173–190.

[27] Jonni Virtema, Jana Hofmann, Bernd Finkbeiner, Juha Kontinen, and Fan Yang. 2020. Linear-Time Temporal Logic with Team Semantics: Expressivity and Complexity. *arXiv:2010.03311 [cs]* (Oct. 2020). arXiv:2010.03311 [cs]

[28] Yu Wang and Miroslav Pajic. 2020. Hyperproperties for Robotics: Motion Planning via HyperLTL. In *IEEE International Conference on Robotics and Automation (ICRA)*. Paris, France, accepted.

[29] Y. Wang, M. Zarei, B. Bonakdarpour, and M. Pajic. 2019. Statistical Verification of Hyperproperties for Cyber-Physical Systems. *ACM Transactions on Embedded Computing systems (TECS)* 18, 5s (2019), 92:1–92:23.

[30] Quanyan Zhu, Linda Bushnell, and Tamer Başar. 2013. Resilient Distributed Control of Multi-Agent Cyber-Physical Systems. In *Control of Cyber-Physical Systems: Workshop Held at Johns Hopkins University, March 2013*, Danielle C. Tarraf (Ed.). Springer International Publishing, Heidelberg, 301–316.