Robotic Planning under Hierarchical Temporal Logic Specifications

Xusheng Luo, Shaojun Xu and Changliu Liu

***Abstract*— Past research into robotic planning with temporal logic specifications, notably Linear Temporal Logic (LTL), was largely based on singular formulas for individual or groups of robots. But with an increase in task complexity, LTL formulas unavoidably grow lengthy, complicating interpretation and specification generation, and straining the computational capacities of the methods applied. In order to maximize the potential of LTL specifications, we capitalized on the intrinsic structure of tasks and introduced a hierarchical structure to LTL specifications. In contrast to the "flat" structure, our hierarchical model has multiple levels of specifications and offers benefits such as greater syntactic brevity, improved interpretability, and more efficient computational planning. To address tasks under this hierarchical temporal logic structure, we formulated a decomposition-based method. Essentially, each specification is first broken down into a range of interrelated temporally sub-tasks. We further examine the temporal relations among the sub-tasks of different specifications within the hierarchy. Subsequently, a Mixed Integer Linear Program is utilized to generate a timed plan for each robot. Our hierarchical LTL specifications were experimentally applied to domains of robotic navigation and manipulation. Numerical simulations illustrated both the enhanced expressive potential of the hierarchical form and the efficacy of the proposed method.**

I. Introduction

Traditionally, robot motion planning primarily involved producing robot paths from a starting to a destination point while avoiding any obstacles [1]. Recently, advanced planning methods have emerged which can manage a more diverse range of tasks beyond the typical point-to-point navigation, encompassing temporal objectives as well. These tasks, which might involve sequencing or coverage [2], data collection [3], intermittent communication [4], and persistent surveillance [5] among others, can be formally represented using languages like Linear Temporal Logic (LTL) [6]. For a comprehensive review of formal specifications and synthesis techniques applicable to robotic systems, refer to [7].

In existing research on LTL planning, LTL tasks are typically assigned to individual robots within a multi-robot team [8], [9], or a global LTL specification is given to the team as a whole [10], [11], capturing the collective behavior of all robots. Regardless of whether the tasks are specified locally or globally, all LTL tasks are presented in a "flat" form, meaning a single LTL formula dictates the behavior of an individual robot or a team of robots. These flat formulas tend to become cumbersome and difficult to interpret for complex tasks. It's been observed that in robotics, tasks often have a loose connection and can be broken down into smaller components. Studies suggest that humans prefer hierarchical task specification, which improves interpretability of planning and execution, making it easier to identify ongoing work and conveniently adjust unfeasible parts without affecting other components [12], [13]. Inspired by this, we propose a highly interpretable hierarchical form of LTL specifications, along with a computationally efficient planner.

Xusheng Luo and Changliu Liu are with Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA (e-mail: {xushengl, cliu6}@andrew.cmu.edu).

Shaojun Xu is with Zhejiang University, China. (e-mail: {sjxu}@zju.edu.cn).

This work was supported by NSF under grant#.

Work done during internship at CMU.

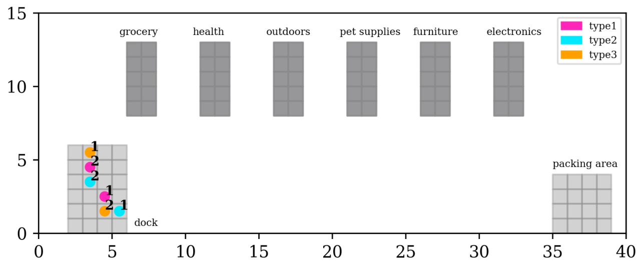


Fig. 1: Topological map of the supermarket. Various types of robots are represented in distinct colors, with each group of robots numbered individually.

Consider the supermarket order picking problem (OPP) [14], where mobile robots are tasked with collecting items from store shelves to fulfill an order. The supermarket's topological map, as shown Figure 1, consists of six sections: grocery, health, outdoor, pet supplies, furniture, and electronics. Robots start at the dock, collect items, and deliver them to the packing area. If an order requires a robot to gather items from the furniture, grocery, and health sections, the robot needs to visit these three sections *in any order*, *subsequently* transport them to the packing area, and *finally* return to the dock. In the traditional "flat" form, this specification would be

$$\begin{aligned}\phi = \Diamond\pi_{\text{furn}} \wedge \Diamond\pi_{\text{groc}} \wedge \Diamond\pi_{\text{heal}} \wedge \Diamond(\pi_{\text{pack}} \wedge \Diamond\pi_{\text{dock}}) \\ \wedge \neg\pi_{\text{pack}} \,\mathcal{U}\, \pi_{\text{furn}} \wedge \neg\pi_{\text{pack}} \,\mathcal{U}\, \pi_{\text{groc}} \wedge \neg\pi_{\text{pack}} \,\mathcal{U}\, \pi_{\text{heal}},\end{aligned}$$

where π_s represents the event of visiting section s. The symbols $\Diamond a$ and $\neg a \,\mathcal{U}\, b$ signify that event a will eventually take place and that event a should not happen until event b happens, respectively. As evident, the formula is cumbersome and difficult to comprehend. We take advantage of the task's inherent hierarchical structure, meaning the robot must *first* collect items *in any order* and *then* deliver them. Thus, the hierarchical form would be

$$\begin{aligned} L_1: \quad \phi &= \Diamond(\phi_1 \wedge \Diamond\phi_2) \\ L_2: \quad \phi_1 &= \Diamond\pi_{\text{furn}} \wedge \Diamond\pi_{\text{groc}} \wedge \Diamond\pi_{\text{heal}} \end{aligned}$$

$$\phi_2 = \Diamond(\pi_{\text{pack}} \wedge \Diamond\pi_{\text{dock}}).$$

The higher level encapsulates the more abstract elements of the task. We underline that the axis distinguishing local from global forms is complementary to the axis distinguishing flat from hierarchical forms. This implies that the hierarchical form can be integrated with both local and global forms. In this work, we employ the global form of hierarchical LTL. Our key contributions can be outlined as follows:

1) A novel hierarchical form of LTL specifications has been introduced, which is more succinct and interpretable than the traditional flat structure;
2) A synthesis method designed to handle hierarchical LTL specifications has been developed, providing computational efficiency and building upon previous work [15];
3) The effectiveness of hierarchical LTL specifications in planning for navigation and manipulation tasks in diverse multi-robot systems has been demonstrated.

II. Related Work

A. Temporal Logic Planning

The field of temporal logic planning has seen rapid expansion in recent years. Much of the research has centered on multi-agent systems, where LTL tasks are either assigned individually to robots within a team [8], [9], [16], [17] or a global LTL specification is used to capture the collective behavior of all robots. In the latter approach, tasks can either be explicitly assigned to each robot, as in [10], [11], [18]–[22], or task allocation is considered without specific assignments, as in [15], [23]–[31]. However, as far as our understanding goes, all LTL formulas, regardless of whether they are assigned locally or globally, are in the flat structure.

As temporal logic formulas are used to tackle complex tasks involving multiple robots and intricate workspaces, they inevitably become lengthy and complicated. Some attempts have been made to simplify this, such as merging multiple atomic propositions into one using logical operators [10], [11], or combining multiple sub-formulas into one formula using logical operators [19], [20], [25], [28]. Despite these efforts, we still categorize these as the flat form due to their lack of depth. Notably, temporal operators, the primary feature distinguishing LTL from propositional logic, only appear on one side, either for the integration of atomic propositions or the combination of sub-formulas. Previous work [15], [24], [28]–[31] has defined atomic propositions involving more than one robot to encapsulate collaborative tasks. In our study, we take this a step further by introducing composite propositions that encompass more than one sub-formula and can be combined using temporal operators.

B. Hierarchical Task Models

Hierarchical reasoning enhances human understanding of the world [12], [13]. In classical AI planning, researchers have crafted various task models reflecting hierarchical structures by employing procedural domain control knowledge [32]. These models have proven to be superior to flat models in terms of interpretability and efficiency, largely due to the significant reduction in the search space for a plan. Hierarchical Task Network (HTN) [33], a commonly used task model in classical AI, exemplifies this. It presents a hierarchy of tasks, each of which can be executed if it's primitive, or broken down into finer sub-tasks if it's complex. The planning process begins with decomposing the initial task network and continues until all compound tasks are decomposed, leading to a solution. The resulting plan comprises a set of primitive tasks applicable to the initial world state. Owing to its expressive capacity, HTN has been implemented in the robotic planning [34], [35]. There are other hierarchical models such as `AND/OR` graphs [36], behavior trees [37], and sequential/parallel graphs [38].

Research combining hierarchical task models with LTL includes studies that use LTL to express temporally extended preferences over tasks and sub-tasks in HTN [39], as well as research into the expressive power of HTN in combination with LTL [40]. However, despite the widespread use of hierarchical task models in classical AI planning, it's intriguing to note the lack of specification hierarchy in temporal logic robotic planning. Our work differs from these studies as we follow an inverse direction; instead of integrating LTL into HTN to express preferences or constraints of tasks, we incorporate HTN into LTL, allowing for hierarchical structures within multiple LTL formulas.

III. Linear temporal logic

Linear Temporal Logic (LTL) is composed of basic statements, referred to as atomic propositions $\mathcal{AP}$, along with boolean operators such as conjunction ($\wedge$) and negation ($\neg$), as well as temporal operators like next ($\bigcirc$) and until ($\mathcal{U}$) [6]. LTL formulas, constructed using these elements over the set of atomic propositions $\mathcal{AP}$, follow the syntax outlined below:

$$\phi := \top \mid \pi \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc \phi \mid \phi_1 \,\mathcal{U}\, \phi_2, \qquad (1)$$

where $\top$ stands for an unconditionally true statement, and π refers to a boolean valued atomic proposition. Other temporal operators can be derived from $\mathcal{U}$, such as $\Diamond\phi$ that implies ϕ will ultimately be true at a future time, and $\Box\phi$ that denotes ϕ will perpetually be true from the current point onwards.

An infinite *word* w over the alphabet $2^{\mathcal{AP}}$, where $\mathcal{AP}$ is the set of atomic propositions, can be denoted as $w = \sigma_0\sigma_1 \ldots \in (2^{\mathcal{AP}})^\omega$, with ω signifying infinite repetition, and $\sigma_k \in 2^{\mathcal{AP}}$ for $\forall k \in \mathbb{N}$. The *language* $\texttt{Words}(\phi)$ is the collection of words that meet the formula ϕ. This means, w belongs to $\texttt{Words}(\phi)$ if and only if $w \models \phi$. The satisfaction relation, represented by $\models$, pairs elements from $(2^{\mathcal{AP}})^\omega \times \phi$.

In this research, our emphasis is on a certain subset of LTL known as syntactically co-safe formulas, or sc-LTL for short [41]. As indicated by [41], it has been established that any LTL formula encompassing only the temporal operators $\Diamond$ and $\mathcal{U}$ and written in positive normal form (where negation is exclusively before atomic propositions) is classified under syntactically co-safe formulas. This category does not include the $\Box$ operator. Sc-LTL formulas can be satisfied by finite sequences followed by any infinite repetitions. This characteristic makes sc-LTL apt for modeling and reasoning about systems with finite durations, such as those found in the robotics field. An LTL formula ϕ can be translated into a Nondeterministic Büchi Automaton (NBA) [42]:

Definition 3.1: (NBA) A *Nondeterministic Büchi Automaton B* is a tuple $B = (\mathcal{Q}, \mathcal{Q}_0, \Sigma, \to_B, \mathcal{Q}_F)$, where $\mathcal{Q}$ is the set of states; $\mathcal{Q}_0 \subseteq \mathcal{Q}$ is a set of initial states; $\Sigma = 2^{\mathcal{AP}}$ is an alphabet; $\to_B \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ is the transition relation; and $\mathcal{Q}_F \subseteq \mathcal{Q}$ is a set of accepting states.

An infinite *run* ρ_B of NBA B over an infinite word w is defined as a sequence $\rho_B = q_0 q_1 q_2 \ldots$, where $q_0 \in \mathcal{Q}_0$ and $(q_k, \sigma_k, q_{k+1}) \in \to_B$, $\forall k \in \mathbb{N}$ with $\sigma_k \in \Sigma$. An infinite run ρ_B is deemed *accepting* if the set of states that appear in ρ_B infinitely often, denoted by $\texttt{Inf}(\rho_B)$, intersects with $\mathcal{Q}_F \neq \varnothing$. In other words, an accepting run involves repeated visits to states associated with accepting conditions. If an LTL formula can be satisfied, there exists an accepting run of NBA B that can be structured in a prefix-suffix manner. The prefix portion is a single traversal path connecting an initial state to an accepting state, while the suffix portion forms an infinite loop around the accepting state. The words σ that lead to an accepting run of NBA B constitute the accepted language of B, denoted as $\mathcal{L}_B$. As established in [6], for any given LTL formula ϕ over a set of atomic propositions $\mathcal{AP}$, there exists a Non-Deterministic Büchi Automaton (NBA) B_ϕ with an alphabet $\Sigma = 2^{\mathcal{AP}}$. The accepted language of B_ϕ, denoted as $\mathcal{L}_{B_\phi}$, corresponds to the set of words accepted by the LTL formula ϕ, denoted as $\texttt{Words}(\phi)$.

IV. Problem Formulation

A. Hierarchical LTL Specifications

We introduce a hierarchical variant of LTL that enables the specification of more intricate tasks than the flat structure, which is rooted in the concept of the *composite proposition*.

Definition 4.1: (Composite proposition) A composite proposition is an LTL formula, excluding the atomic proposition, that conforms to the grammar rules in (1).

The truth of a composite proposition is determined by the amalgamation of underlying atomic propositions. In terms of syntax, composite propositions enhance atomic propositions by integrating temporal and logic operators. Specifications such as $\bigcirc\phi$, $\phi_1 \,\mathcal{U}\, \phi_2$, and $\Diamond\phi$ serve as instances of composite propositions. Before we define hierarchical LTL formally, we utilize the following example to help explain the concept.

Example 1: (Order Picking Problem) We consider three types of robots, each possessing different capabilities, and there are two robots of each type; refer to Figure 1. The specification for one instance of an OPP is as follows:

1) *Initially*, a robot of type 1 proceeds to the furniture section and awaits *until* the arrival of a type 3 robot to assist with the movement of a large piece of furniture; *following* this, it gathers items from the outdoor and pet sections *in no particular sequence*.
2) A type 2 robot should *initiate* its task by gathering items from the health section, and *only afterwards* does it move to the grocery section, ensuring it doesn't visit the grocery section *before* visiting the health section.
3) A type 3 robot *begins* by gathering items from the electronics section and *subsequently* moves to the furniture section to assist the type 1 robot.
4) *After* the items are delivered to the packing area, all robots *eventually* return to the dock.

The hierarchical OPP specification is structured as follows:

$$
\boxed{\text{Task 1}}\quad
\begin{aligned}
L_1:&\quad \phi_1^1 = \Diamond\phi_1^2 \wedge \Diamond\phi_2^2\\
L_2:&\quad \phi_1^2 = \Diamond(\phi_1^3 \wedge \Diamond\phi_2^3 \wedge \Diamond(\phi_3^3 \wedge \Diamond\phi_4^3))\\
&\quad \phi_2^2 = \Diamond(\phi_5^3 \wedge \Diamond\phi_6^3)\\
L_3:&\quad \phi_1^3 = \Diamond(\pi_{\text{furn}}^{1,1} \wedge \bigcirc(\pi_{\text{furn}}^{1,1} \,\mathcal{U}\, \pi_{\text{furn}}^{3,3}))\\
&\quad \phi_2^3 = \Diamond(\pi_{\text{pack}}^{3,3} \wedge \Diamond\pi_{\text{dock}}^{3,3})\\
&\quad \phi_3^3 = \Diamond\pi_{\text{outd}}^{1,1} \wedge \Diamond\pi_{\text{pet}}^{1,1}\\
&\quad \phi_4^3 = \Diamond(\pi_{\text{pack}}^{1,1} \wedge \Diamond\pi_{\text{dock}}^{1,1})\\
&\quad \phi_5^3 = \Diamond\pi_{\text{heal}}^{2,2} \wedge \Diamond\pi_{\text{groc}}^{2,2} \wedge \neg\pi_{\text{groc}}^{2,2} \,\mathcal{U}\, \pi_{\text{heal}}^{2,2}\\
&\quad \phi_6^3 = \Diamond(\pi_{\text{pack}}^{2,2} \wedge \Diamond\pi_{\text{dock}}^{2,2})
\end{aligned}
\tag{2}
$$

The symbols π_s^t stand for atomic propositions signifying that a robot of type t is required to collect items from section s. If two atomic propositions have the same second superscript, they need to be executed by the same robot. For instance, $\pi_{\text{furn}}^{1,1}, \pi_{\text{outd}}^{1,1}$ and $\pi_{\text{pet}}^{1,1}$ should be completed by the same robot of type 1, as should $\pi_{\text{heal}}^{2,2}, \pi_{\text{groc}}^{2,2}$. Levels are represented by $L_1 \sim L_3$ and specifications at various levels are denoted by ϕ_i^k. Looking at ϕ_2^2, ϕ_5^3 indicates that a single type 2 robot should initially visit the health section, then proceed to the grocery section; ϕ_5^3 dictates that the same robot should carry items to the packing area and eventually return to the dock. ϕ_2^2 arranges the order of these events. A flat LTL specification of the OPP task from Example 1 can be seen in (8) in Appendix I. It's important to note that the flat and hierarchical versions of the same task are not strictly equivalent in terms of the set of accepted words, given the challenge in task specification in the flat form. $\square$

The hierarchical LTL specifications are divided into several levels, denoted by K, each of which can include multiple specifications. From the top level down, we denote the i-th specification at level $k \in \{1, \ldots, K\}$ as ϕ_i^k. Except for the specifications on the top level, for any specification ϕ_i^k at level $k > 1$, there is a unique composite proposition that is solely present in one specification ϕ_j^{k-1}, such that the satisfaction of ϕ_i^k guarantees the satisfaction of the corresponding composite proposition in ϕ_j^{k-1}. We also slightly bend the notation to use ϕ_i^k to denote this composite proposition at level $k-1$. In this manner, ϕ_i^k represents not only the i-th specification at level k, but also the composite proposition at level $k-1$. In this work, when ϕ_i^k appears in a certain formula, we consider it as a composite proposition. On the other hand, when it appears as a standalone formula, we refer to it as a specification.

Consider a level k that isn't the bottommost one, we denote a composite proposition as $\phi_i^k = \Gamma_{j \in \mathcal{I}_\downarrow}(\phi_j^{k+1})$, where $\mathcal{I}_\downarrow$ is the collection of specifications at level $k+1$ that serve to break down proposition ϕ_i^k, and Γ denotes the operation that assembles specifications from lower levels in a specific manner to create the composite proposition. Similarly, let $\mathcal{I}^{\uparrow}(\phi_i^k)$ denote the set of specifications at levels $k-1$ where it appears as a composite proposition.

Example 1: continued (Hierarchical OPP) $\mathcal{I}^{\downarrow}(\phi_1^1) = \{\phi_1^2, \phi_2^2\}$; $\mathcal{I}^{\downarrow}(\phi_1^2) = \{\phi_1^3, \phi_2^3, \phi_3^3, \phi_4^3\}$; $\mathcal{I}^{\uparrow}(\phi_1^2) = \{\phi_1^1\}$; $\mathcal{I}^{\uparrow}(\phi_2^2) = \{\phi_1^1\}$; $\mathcal{I}^{\uparrow}(\phi_1^3) = \{\phi_1^2\}$; $\mathcal{I}^{\uparrow}(\phi_2^3) = \{\phi_1^2\}$.

Definition 4.2: (Hierarchical LTL) A hierarchical linear temporal logic specification, denoted by $\{\phi_i^k\}_{k=1}^K$, includes K levels such that each composite proposition at level $k \in \{1, \ldots, K-1\}$ is constructed from specifications at level $k+1$, i.e., $\phi_i^k = \Gamma_{j \in \mathcal{I}_{\downarrow}}(\phi_j^{k+1})$.

This hierarchical structure for task representation encapsulates different degrees of abstraction. More abstract specifications are involved at higher levels, whereas detailed implementations are concentrated at lower levels. We observe that hierarchical LTL has a representational edge over its flat counterpart, which is detailed in the simulation section.

The requirements on valid hierarchical LTL specifications are listed as follows:

1) All propositions at the lowest level must be atomic propositions, though atomic propositions can also be present at non-bottom levels;
2) Every specification at level k (excluding the highest one), should appear as composite propositions in one and only one specification at the level $k-1$;
3) A composite proposition cannot be linked with another composite/atomic propositions by the logical operator $\wedge$ in any specification;
4) Any feasible plan that fulfills a certain specification should not invalidate other specifications.

Condition 2) implies that $\mathcal{I}^{\uparrow}(\phi_i^k)$ yields a singleton set. Condition 3) allows for the simultaneous fulfillment of atomic propositions only. This is because the truth value of a composite proposition depends on multiple atomic propositions and the time at which they become true. In the context of robotics, it isn't logical to assert that a composite proposition becomes true concurrently with another composite or atomic proposition. As for condition 4), "does not invalidate other specifications" implies that it's possible to supplement any viable plan of a particular specification with an additional plan to meet other specifications. For instance, the third level specification ϕ_3^3 in (2), regardless of the order in which the outdoor or pet section is visited, doesn't affect the truth value of other specifications. That is, higher level specifications are concerned primarily with whether a certain composite proposition is fulfilled, rather than how it is fulfilled.

B. Problem Formulation

In the context of applying hierarchical LTL specifications to the field of robotics, we make the following assumption:

Assumption 4.3: For every specification ϕ_i^k, the atomic propositions generated by the robots' initial states should activate either the self-transition or an outgoing transition of at least one initial Büchi state in the corresponding NBA B_i^k.

If Assumption 4.3 does not hold for a specification, no transitions will be initiated within its NBA. Consequently, the task becomes unachievable.

Problem 1: Assuming that Assumption 4.3 is valid. Considering a domain of robotic navigation or manipulation, given a group of m robots and a valid hierarchical LTL specification ϕ, generate an optimal plan that fulfills the task.

Remark 4.4: In this work, our focus is on minimizing the overall time. The optimization of other aspects of the plan, such as energy consumption, is a subject for future work.

V. Decomposition-based Planning

The proposed approach is an extension of our prior research, as outlined in [15]. This previous work decomposes the NBA of a given LTL specification and infers the temporal relationships between sub-tasks (which will be defined shortly). In this study, we first establish the temporal connections among sub-tasks for each specification, then deduce the temporal relations between sub-tasks across different specifications. Following this, we implement a Mixed Integer Linear Programming (MILP) model to distribute the sub-tasks among the robots. The robots then carry out the tasks using domain-specific low-level planning mechanisms.

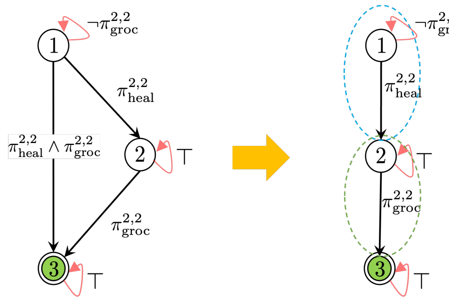
We start by presenting the necessary definitions and notations related to NBA, as detailed in [15]. It's worth noting that the NBA, as defined in Definition 3.1, can be perceived as a graph. For the sake of simplicity, we'll refer to the NBA as the graph $\mathcal{A}_\phi$ in the remainder of this paper. When discussing $\mathcal{A}_\phi$'s edges, we don't consider self-loops as they can be represented by vertices. The propositional formula γ associated with a transition $v_1 \xrightarrow{\gamma} v_2$ in the NBA $\mathcal{A}_\phi$ is called a *vertex label* if $v_1 = v_2$, and an *edge label* if not. We denote the mappings of a vertex and an edge to their respective labels using the functions $\gamma : \mathcal{V} \to \Sigma$ and $\gamma : \mathcal{V} \times \mathcal{V} \to \Sigma$, respectively. We then define an edge-induced *sub-task* as a series of actions that robots must perform to trigger a transition in the NBA.

One important note: any edge whose label includes a clause in which the $\wedge$-connected sub-formula does not appear in the specification should be eliminated. This can greatly decrease the size of the NBA.

Definition 5.1: (Edge-induced sub-task [15]) In the NBA $\mathcal{A}_\phi$, given an edge (v_1, v_2), an edge-induced sub-task is characterized by the edge label $\gamma(v_1, v_2)$ and the label of the starting vertex $\gamma(v_1)$.

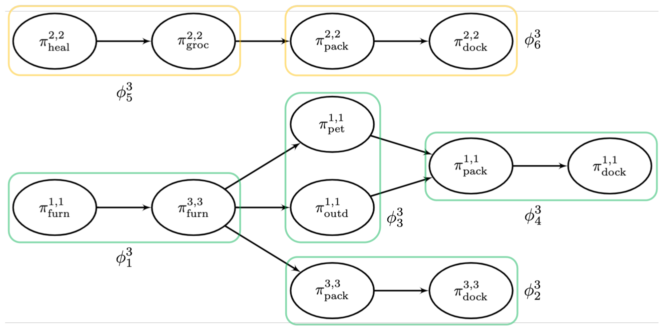
Edge-induced sub-tasks can be viewed as a generalized reach-avoid tasks. In these tasks, robots are required to perform a specific action to activate the edge label (the "reach" component), while simultaneously maintaining the fulfillment of the starting vertex labels throughout the execution process (the "avoid" component).

Definition 5.2: (Atomic and composite sub-task) In the NBA $\mathcal{A}_\phi$, a sub-task (v_1, v_2) is classified as an atomic subtask if the edge label $\gamma(v_1, v_2)$ consists solely of atomic



$\neg\pi^{2,2}_{\text{groc}}$
$\pi^{2,2}_{\text{heal}}$
$\pi^{2,2}_{\text{heal}} \wedge \pi^{2,2}_{\text{groc}}$
$\top$
$\pi^{2,2}_{\text{groc}}$
1
2
3

Fig. 2: The left side represents the NBA of specification ϕ_5^3 at the third level of the OPP task as referenced in equation (2), whereas the right side is the outcome from work [15]. It extracted two sub-tasks (v_1, v_2) and (v_2, v_3), highlighted by ellipses. The sub-task (v_1, v_2) implies that a robot of type 2 is required to visit the health section (as indicated by the edge label), and until that point, it should avoid entering the grocery section (as depicted by the vertex label). These two sub-tasks follow each other in sequence.



$\pi^{2,2}_{\text{heal}}$
$\pi^{2,2}_{\text{groc}}$
$\pi^{2,2}_{\text{pack}}$
$\pi^{2,2}_{\text{dock}}$
ϕ^3_6
ϕ^3_5
$\pi^{1,1}_{\text{pet}}$
$\pi^{1,1}_{\text{furn}}$
$\pi^{3,3}_{\text{furn}}$
$\pi^{1,1}_{\text{outd}}$
$\pi^{1,1}_{\text{pack}}$
$\pi^{1,1}_{\text{dock}}$
ϕ^3_3
ϕ^3_4
ϕ^3_1
$\pi^{3,3}_{\text{pack}}$
$\pi^{3,3}_{\text{dock}}$
ϕ^3_2

Fig. 3: Task network of task 1; see Equation (2). The symbols within each node correspond to the edge label, while the vertex labels have been omitted for clarity of presentation. Nodes that belong to the same specification are grouped together within a box.

propositions. Conversely, if it contains anything other than atomic propositions, it's considered a composite sub-task.

Example 1: continued (Sub-tasks) The corresponding NBA of specification ϕ_5^3 and edge-induced sub-tasks are shown in Figure 2.

A. Generate Temporal Relations Across Specifications

The work [15] takes a given LTL specification, extracts edge-induced sub-tasks from its NBA, and determines the temporal relationships between these sub-tasks; see Figure 2 for the specification ϕ_5^3. As various LTL specifications are arranged in a hierarchical manner, we process each LTL specification separately. Afterward, we deduce the temporal relationships between sub-tasks derived from different LTL specifications. The ultimate aim is to establish a task network represented by a Directed Acyclic Graph (DAG). In this graph, every node signifies an atomic sub-task, and an edge is present between a pair of atomic sub-tasks if there are existing precedence relationships between them. The task network of task 1 is shown in Figure 3.

Let ϕ^k represent the set of specifications at the k-th level, and let $|\phi^k|$ signify its cardinality. In what follows, we will denote the edge-induced sub-task as $e = (v_1, v_2)$, and use $[\![\phi]\!], [\![\phi]\!]_a$, and $[\![\phi]\!]_c$ to represent the sets of sub-tasks, atomic sub-tasks, and composite sub-tasks for the given specification ϕ, respectively. For any two distinct sub-tasks $e, e' \in [\![\phi_i^k]\!]$, we define $e \bowtie e'$, where $\bowtie \in \{\prec, \succ, \|\}$, to indicate that sub-task e needs to be completed before ($\prec$), after ($\succ$), or independent ($\|$) of e'. We denote the set of composite propositions in the hierarchical LTL by Φ, and a general composite proposition by $\phi \in \Phi$. The subsequent assumption restricts the relationships between sub-tasks of two composite propositions.

Assumption 5.3: (Temporal relation inheritance) Sub-tasks inherit the temporal relations of their respective composite propositions. That is, for any pair of composite propositions $\phi_i^k, \phi_{i'}^{k'}$, if $\phi_i^k \bowtie \phi_{i'}^{k'}$, then this relationship holds for all sub-tasks, i.e., $e \bowtie e', \forall e \in [\![\phi_i^k]\!], \forall e' \in [\![\phi_{i'}^{k'}]\!]$.

The algorithm for constructing the DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ from hierarchical LTL specifications is outlined in Algorithm 1. The first step in constructing the DAG involves iterating over each specification and adding an edge to the graph if a precedence relation exists between its two atomic sub-tasks, as seen in lines 1-7.

For a composite proposition ϕ_i^k, its *predecessor* is the composite proposition that encapsulates it at a higher level, which can be identified through $\mathcal{I}^\uparrow(\phi_i^k)$. According to condition 2), its predecessor is unique. Its sequence of predecessors, represented by $\mathcal{S}(\phi_i^k)$, can be determined by backtracking up to the first level. Note that a composite proposition can be its own predecessor. When provided with two sequences $\mathcal{S}(\phi_i^k)$ and $\mathcal{S}(\phi_{i'}^{k'})$, we refer to a proposition that appears in both sequences and is the closest to them in terms of levels as the *minimal common predecessor*, denoted by ϕ_{min}. The temporal relations between two composite propositions can be determined within their minimal common predecessor. This is dependent on whether one composite proposition is the predecessor of the other:

(a) If this is the case, one of them is considered the minimal common predecessor. Assume that ϕ_i^k is the predecessor of $\phi_{i'}^{k'}$. It's plausible that $\phi_{i'}^{k'}$ is not found in ϕ_i^k due to the presence of multiple levels between them. We first locate the composite proposition ϕ that serves as the predecessor of $\phi_{i'}^{k'}$ and is also included in ϕ_i^k. Then, we identify the sub-task e_ϕ in ϕ_i^k whose edge label includes ϕ without a preceding $\neg$, that is, the sub-task that necessitates the satisfaction of ϕ, which in turn necessitates the satisfaction of $\phi_{i'}^{k'}$. According to Assumption 5.3, for each atomic sub-task in $[\![\phi_i^k]\!]_a$, its temporal relationship with the atomic sub-tasks in $[\![\phi_{i'}^{k'}]\!]_a$ matches its relationship with the composite sub-task e_ϕ, as demonstrated in lines 9-16.

(b) If not, similar to case (a), we identify the composite propositions ϕ and ϕ' that are predecessors of ϕ_i^k and $\phi_{i'}^{k'}$, respectively, and are also contained in ϕ_{min}. Next, we locate the sub-tasks e_ϕ and $e_{\phi'}$ in ϕ_{min} whose edge labels

Algorithm 1: Construct DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

```
Input: Hierarchical LTL specifications $\{\phi_i^k\}_{k=1}^K$
Output: DAG $\mathcal{G}$ of atomic sub-tasks
1  for $k \in \{K, \ldots, 1\}$ do
2      for $i \in \{1, \ldots, |\phi^k|\}$ do
           ; ▷ Two atomic sub-tasks for the same specification
3          for $e, e' \in [\![\phi_i^k]\!]_a$ do
4              if $e \prec e'$ then
5                  $\mathcal{E} = \mathcal{E} \cup (e, e')$;
6              else if $e \succ e'$ then
7                  $\mathcal{E} = \mathcal{E} \cup (e', e)$ ;
   ;          ▷ Two composite propositions
8  for $\phi_i^k, \phi_{i'}^{k'} \in \Phi$ do
9      if one is the predecessor of the other then
           ;        ▷ $\phi_i^k$ is predecessor of $\phi_{i'}^{k'}$
10         Find sub-task $e_\phi$ in $\phi_i^k$;
11         for $e \in [\![\phi_i^k]\!]_a$ do
12             for $e' \in [\![\phi_{i'}^{k'}]\!]_a$ do
13                 if $e \prec e_\phi$ then
14                     $\mathcal{E} = \mathcal{E} \cup (e, e')$;
15                 else if $e \succ e_\phi$ then
16                     $\mathcal{E} = \mathcal{E} \cup (e', e)$ ;
17     else
18         Find sub-tasks $e_\phi$ and $e_{\phi'}$ in $\phi_{\text{min}}$;
19         for $e \in [\![\phi_i^k]\!]_a$ do
20             for $e' \in [\![\phi_{i'}^{k'}]\!]_a$ do
21                 if $e_\phi \prec e_{\phi'}$ then
22                     $\mathcal{E} = \mathcal{E} \cup (e, e')$;
23                 else if $e_\phi \succ e_{\phi'}$ then
24                     $\mathcal{E} = \mathcal{E} \cup (e', e)$;
```

incorporate ϕ and ϕ', respectively, without a preceding $\neg$. The temporal relationship between ϕ_i^k and $\phi_{i'}^{k'}$ mirrors the relationship between the corresponding sub-tasks e_ϕ and $e_{\phi'}$ within ϕ_{min}. Based on Assumption 5.3, the temporal relations between atomic sub-tasks found in $[\![\phi_i^k]\!]_a$ and $[\![\phi_{i'}^{k'}]\!]_a$ can be established, as seen in lines 18-24.

Example 1: continued (Temporal relations) In terms of the specifications ϕ_1^3 and ϕ_2^3, sequences of predecessors are $\mathcal{S}(\phi_1^3) = \{\phi_1^3, \phi_1^2, \phi_1^1\}, \mathcal{S}(\phi_2^3) = \{\phi_2^3, \phi_1^2, \phi_1^1\}$, minimal common predecessor is $\phi_{\text{min}} = \phi_1^2$, predecessors in ϕ_{min} are ϕ_1^3 and ϕ_2^3 themselves. Given that $\phi_1^3 \prec \phi_2^3$ in ϕ_1^2, the temporal relationships between atomic sub-tasks from the two given specifications become $\pi_{\text{furn}}^{1,1} \prec \pi_{\text{pack}}^{3,3}, \pi_{\text{furn}}^{1,1} \prec \pi_{\text{dock}}^{3,3}, \pi_{\text{furn}}^{3,3} \prec \pi_{\text{pack}}^{3,3}, \pi_{\text{furn}}^{3,3} \prec \pi_{\text{dock}}^{3,3}$. Note that these sub-tasks are denoted by their respective edge labels.

B. Formulate MILP to Allocate Tasks

Given the graph $\mathcal{G}$, we adjust the MILP formulation from [15], to handle multiple specifications. For a more detailed explanation, we direct readers to Appendix A in [15]. All amendments are made to incorporate specification-level variables, which imposes specification-level constraints, thereby controlling sub-task-level constraints.

For a specification ϕ, let's consider the propositional formula γ associated with a transition $v_1 \xrightarrow{\gamma} v_2$ in the NBA $\mathcal{A}_\phi$. This formula is in *disjunctive normal form* (DNF), i.e., $\gamma = \bigvee_{p \in \mathcal{P}} \bigwedge_{q \in \mathcal{Q}_p} (\neg)\pi$, where the negation operator only precedes the atomic propositions and $\mathcal{P}$ and $\mathcal{Q}_p$ are suitable index sets. It's important to note that every propositional formula has an equivalent DNF form as per [6]. We define $\mathcal{C}_p^\gamma = \bigwedge_{q \in \mathcal{Q}_p} (\neg)\pi$ as the p-th *clause* of γ, and we use $\mathsf{cls}(\gamma)$ to represent the set of clauses within γ.

In [15], for a specification ϕ, there is an equality constraint for each propositional formula γ, which necessitates that one and only one clause should be true to satisfy γ. This may not hold true for hierarchical cases. Take, for instance, a propositional formula where two composite propositions are linked by the logical operator "OR", i.e., $\gamma = \phi_i^k \vee \phi_j^k$. γ will be deemed true as long as one of the composite propositions is fulfilled, indicating that it is not mandatory to satisfy all clauses included in the other composite proposition. To account for these constraints, we introduce a binary variable $b_{\phi_i^k}$ to signify the truth of specification ϕ_i^k. For each propositional formula γ, let the set Φ_γ gather all composite propositions in γ that are interconnected by $\vee$. The constraint that one and only one composite proposition in Φ_γ is true can be encoded by:

$$\sum_{\phi_i^k \in \Phi_\gamma} b_{\phi_i^k} = 1. \tag{3}$$

By default, $b_{\phi_i^k}$ is set to 1 if ϕ_i^k does not co-occur with other composite propositions linked by $\vee$ operators. For a propositional formula γ of specification ϕ_i^k, we define a binary variable b_p to signify the truth of the p-th clause. Consequently, the constraint that γ should only be true if ϕ_i^k is true can be represented by:

$$\sum_{p \in \mathcal{P}} b_p = b_{\phi_i^k}, \tag{4}$$

which modifies the equality constraint (9) in [15], which is expressed as $\sum_{p \in \mathcal{P}} b_p = 1$.

Another similar adjustment applies to the inequality constraint associated with each propositional formula in ϕ_i^k, which doesn't need to be activated if the related specification is false. To represent this, the key idea is that inequality constraints are inherently fulfilled if the specification ϕ_i^k is not activated. Specifically, each inequality constraint in [15] takes the form of a general linear inequality $g(x) \le 0$, where x are the decision variables of MILP. The following ensures it's satisfied trivially if ϕ_i^k is not enabled:

$$g(x) \le M(1 - b_{\phi_i^k}), \tag{5}$$

which is activated if $b_{\phi_i^k} = 1$. We implement such modifications to constraints (15), (17), (18), (19b) and (23) in [15].

Remark 5.4: After the assignment of tasks, each one will be carried out with domain-specific controllers. For the running navigation task, the Generalized Multi-Robot Path Planner in [15] creates the robot paths, which are then examined to check whether they meet the LTL specifications. If the satisfaction check is unsuccessful, no solution is provided. In this case, the proposed method is sound. Note that [15] is complete for a broad range of LTL specifications. By building upon it, our approach becomes feasible for typical robotic tasks, which is validated in the subsequent simulation section.

VI. Numerical Experiments

In this section, we apply the proposed method to the realms of navigation and manipulation, employing Python 3.10.12 on a computer with 3.5 GHz Apple M2 Pro and 16G RAM. The Big-M based MILP is resolved using the Gurobi solver [43], where the big-M constant is set to 10^5.

A. Navigation Task

We continue using the supermarket environment, with the robot's starting locations being arbitrarily selected within the dock. A robot is considered to have visited a section if it approaches the aisle that lies adjacent to any side of a shelf. Two additional tasks are also examined below. The extracted task networks are depicted in Figures 5 and 6. A video of synthesized plans for all 3 tasks are accessible via https://youtu.be/fh5i_RphhMA.

$$\boxed{\text{Task 2}} \quad \begin{aligned} L_1: \quad & \phi_1^1 = \Diamond(\phi_1^2 \wedge \Diamond(\phi_2^2 \wedge \Diamond(\phi_3^2 \wedge \Diamond\phi_4^2))) \\ L_2: \quad & \phi_1^2 = \Diamond\pi_{\text{furn}}^{1,1} \wedge \Diamond\pi_{\text{outd}}^{1,1} \\ & \phi_2^2 = \Diamond\pi_{\text{heal}}^{1,1} \wedge \Diamond\pi_{\text{groc}}^{1,1} \\ & \phi_3^2 = \Diamond\pi_{\text{elec}}^{1,1} \wedge \Diamond\pi_{\text{pet}}^{1,1} \\ & \phi_4^2 = \Diamond(\pi_{\text{pack}}^{1,1} \wedge \Diamond\pi_{\text{dock}}^{1,1}) \end{aligned} \tag{6}$$

$$\boxed{\text{Task 3}} \quad \begin{aligned} L_1: \quad & \phi_1^1 = \Diamond(\phi_1^2 \wedge \Diamond\phi_2^2) \wedge (\Diamond\phi_3^2 \vee \Diamond\phi_4^2) \\ L_2: \quad & \phi_1^2 = \Diamond\pi_{\text{heal}}^{1,1} \wedge \Diamond\pi_{\text{groc}}^{1,1} \wedge \Diamond\pi_{\text{elec}}^{1,1} \wedge \Diamond\pi_{\text{pet}}^{1,1} \\ & \phi_2^2 = \Diamond(\pi_{\text{pack}}^{1,1} \wedge \Diamond\pi_{\text{dock}}^{1,1}) \\ & \phi_3^2 = \Diamond(\pi_{\text{outd}}^{2,2} \wedge \Diamond(\pi_{\text{pack}}^{2,2} \wedge \Diamond\pi_{\text{dock}}^{2,2})) \\ & \phi_4^2 = \Diamond(\pi_{\text{outd}}^{3,3} \wedge \Diamond(\pi_{\text{pack}}^{3,3} \wedge \Diamond\pi_{\text{dock}}^{3,3})) \end{aligned} \tag{7}$$

The *length* of a LTL specification is quantified by the number of operators it contains, which can include both logical and temporal operators [6]. This metric can serve as an indicator of the challenge associated with both formulating and interpreting the specification. Flat versions of tasks 2 and 3 can be found in Appendix I, identified as (9) and (10) respectively. For each task, we compare between the length of the LTL and the size of the NBA, taking into account both the flat and hierarchical forms.[1] In the case of the hierarchical form, the total is the sum of all the specifications. The hierarchical form is addressed with our proposed method, while the flat form is dealt with by applying the method from [15] for comparison. The initial locations for the robots are randomly sampled within the dock, and the runtimes and plan horizons are averaged over 20 runs. The statistical outcomes are displayed in Table I. The hierarchical form of LTL specifications leads to more concise formulas in length and NBAs that are reduced in size with fewer nodes and edges. The reduction in complexity leads to decreased runtimes and costs, particularly for task 1, where it notably minimizes both the runtimes and planning horizons.

[1]These NBAs are built using LTL2BA developed by [44].

task	l_{flat}	l_{hier}	$\mathcal{A}_{\text{flat}}$	$\mathcal{A}_{\text{hier}}$
1	51	35	(387, 13862)	(33, 45)
2	45	19	(12, 63)	(20, 28)
3	35	27	(113, 1891)	(35, 101)

task	t_{flat}	t_{hier}	c_{flat}	c_{hier}
1	126.1±2.7	18.4±0.8	289.3±3.0	237.5±3.3
2	30.8±0.5	24.8±0.4	115.6±1.9	114.8±1.6
3	25.8±1.8	21.4±0.5	148.1±2.1	147.7±2.1

TABLE I: Comparative results of two forms of LTL specifications. The length of LTL specifications is represented by l_{flat} and l_{hier}, while the size of the NBA is denoted by $\mathcal{A}_{\text{flat}}$ and $\mathcal{A}_{\text{hier}}$, specifying the number of nodes first, followed by the number of edges. The runtimes taken to find the solutions are indicated by t_{flat} and t_{hier}, and the plan horizons of the solutions are indicated by c_{flat} and c_{hier}.

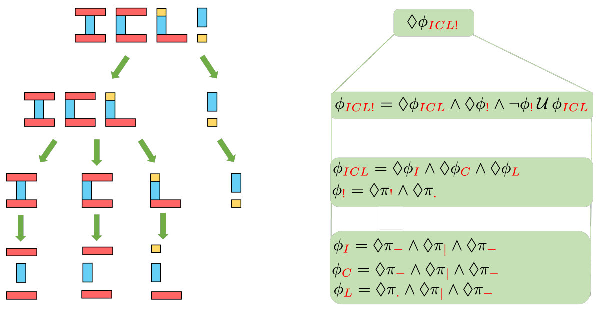

$\Diamond\phi_{ICL!}$
$\phi_{ICL!} = \Diamond\phi_{ICL} \wedge \Diamond\phi_{!} \wedge \neg\phi_{!}\,\mathcal{U}\,\phi_{ICL}$
$\phi_{ICL} = \Diamond\phi_I \wedge \Diamond\phi_C \wedge \Diamond\phi_L$
$\phi_{!} = \Diamond\pi_{\text{!`}} \wedge \Diamond\pi_{.}$
$\phi_I = \Diamond\pi_{-} \wedge \Diamond\pi_{|} \wedge \Diamond\pi_{-}$
$\phi_C = \Diamond\pi_{-} \wedge \Diamond\pi_{|} \wedge \Diamond\pi_{-}$
$\phi_L = \Diamond\pi_{.} \wedge \Diamond\pi_{|} \wedge \Diamond\pi_{-}$

Fig. 4: A 4-level hierarchical structure used in LEGO assembly. While atomic propositions π_{-} occur twice in ϕ_I, they are not the same due to differing block configurations.

B. Manipulation Task

Consider a scenario involving human-robot collaboration where the objective is to assemble LEGO blocks into a formation that spells "ICL!", as illustrated in Figure 4. Different colored blocks represent distinct types, and either a human or robot can assemble each block. The task necessitates assembling the "ICL" letters *initially, followed* by "!". Let ϕ_X represent the composite proposition of assembling module X. The task can be succinctly defined by the LTL formula $\Diamond\phi_{ICL} \wedge \Diamond\phi_{!} \wedge \neg\phi_{!}\,\mathcal{U}\,\phi_{ICL}$. The hierarchy of the LEGO assembly process is depicted in Figure 4. At the first level, the overall task is to assemble "ICL!", specified by $\Diamond\phi_{ICL!}$. At the second level, the task is broken down with temporal constraints. At the third level, the task implies that the "ICL" letters can be assembled in any sequence. Similarly, the upper and lower parts of the "!" symbol can also be assembled without following a particular order. The most basic steps to assemble each letter are stated at the lowest level. A video demonstrating the solution in the simulator can be found at the provided Youtube link.

To highlight the complexity difference between the hierarchical and flat versions, consider assembling the three letters "ICL". The hierarchical version comprises 4 specifications: $\phi_{ICL}, \phi_I, \phi_C$, and ϕ_L. The corresponding NBAs contain 32 states and 104 transitions in total (8 states and 26 transitions per specification). However, the NBA of the flat version, which is expressed as $\Diamond\phi_{\_} \wedge \Diamond\phi_{|} \ldots \wedge \Diamond\phi_{\_}$, contains a significantly larger number of 512 states and 19682 transitions. Similarly, to encapsulate the constraint $\neg\phi_{!}\,\mathcal{U}\,\phi_{ICL}$, it needs to be expressed as $\neg\phi_{.}\,\mathcal{U}\,\phi_{\_} \wedge \ldots \wedge \neg\phi_{.}\,\mathcal{U}\,\phi_{\_}$ for the lower part of the "!", which has to be repeated for the 9 basic operations required to assemble "ICL". The same process is then repeated for the upper part.

VII. Limitations and Discussions

While the hierarchical LTL introduced in this work has proven effective in two robotic applications, there are still several areas that require additional investigation. Firstly, the automatic transforming a basic LTL into a hierarchical structure is an interesting problem to tackle. Secondly, our present approach is efficient yet lacks in comprehensiveness; therefore, devising a method that ensures both completeness and optimality presents a compelling objective. Lastly, future research could aim to lift the requirements for a valid hierarchical LTL, especially concerning points 3) and 4).

VIII. Conclusions

In our work, we introduced a hierarchical variant of LTL specifications, which offers benefits in terms of conciseness, clarity, and computational efficiency. We also put forward a decomposition-based approach to tackle robotic navigation and manipulation tasks represented using these hierarchical LTL specifications.

References

[1] S. M. LaValle, Planning algorithms. Cambridge university press, 2006.

[2] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in IEEE International Conference on Robotics and Automation (ICRA), Barcelona, Spain, 2005, pp. 2020–2025.

[3] M. Guo and M. M. Zavlanos, "Distributed data gathering with buffer constraints and intermittent communication," in 2017 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2017, pp. 279–284.

[4] Y. Kantaros and M. M. Zavlanos, "Distributed intermittent connectivity control of mobile robot networks," IEEE Transactions on Automatic Control, vol. 62, no. 7, pp. 3109–3121, 2017.

[5] K. Leahy, D. Zhou, C.-I. Vasile, K. Oikonomopoulos, M. Schwager, and C. Belta, "Persistent surveillance for unmanned aerial vehicles subject to charging and temporal logic constraints," Autonomous Robots, vol. 40, no. 8, pp. 1363–1378, 2016.

[6] C. Baier and J.-P. Katoen, Principles of model checking. MIT press Cambridge, 2008.

[7] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, and M. Fisher, "Formal specification and verification of autonomous robotic systems: A survey," ACM Computing Surveys (CSUR), vol. 52, no. 5, pp. 1–41, 2019.

[8] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local LTL specifications," The International Journal of Robotics Research, vol. 34, no. 2, pp. 218–235, 2015.

[9] J. Tumova and D. V. Dimarogonas, "Multi-agent planning under local LTL specifications and event-based synchronization," Automatica, vol. 70, pp. 239–248, 2016.

[10] Y. Kantaros and M. M. Zavlanos, "Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems," The International Journal of Robotics Research, vol. 39, no. 7, pp. 812–836, 2020.

[11] X. Luo, Y. Kantaros, and M. M. Zavlanos, "An abstraction-free method for multirobot temporal logic optimal control synthesis," IEEE Transactions on Robotics, vol. 37, no. 5, pp. 1487–1507, 2021.

[12] J. B. Tenenbaum, C. Kemp, T. L. Griffiths, and N. D. Goodman, "How to grow a mind: Statistics, structure, and abstraction," science, vol. 331, no. 6022, pp. 1279–1285, 2011.

[13] C. Kemp, A. Perfors, and J. B. Tenenbaum, "Learning overhypotheses with hierarchical bayesian models," Developmental science, vol. 10, no. 3, pp. 307–321, 2007.

[14] L. Pansart, N. Catusse, and H. Cambazard, "Exact algorithms for the order picking problem," Computers & Operations Research, vol. 100, pp. 117–127, 2018.

[15] X. Luo and M. M. Zavlanos, "Temporal logic task allocation in heterogeneous multirobot systems," IEEE Transactions on Robotics, vol. 38, no. 6, pp. 3602–3621, 2022.

[16] P. Yu and D. V. Dimarogonas, "Distributed motion coordination for multirobot systems under ltl specifications," IEEE Transactions on Robotics, vol. 38, no. 2, pp. 1047–1062, 2021.

[17] R. Bai, R. Zheng, Y. Xu, M. Liu, and S. Zhang, "Hierarchical multi-robot strategies synthesis and optimization under individual and collaborative temporal logic specifications," Robotics and Autonomous Systems, vol. 153, p. 104085, 2022.

[18] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multi-agent motion tasks based on LTL specifications," in 43rd IEEE Conference on Decision and Control (CDC), vol. 1, The Bahamas, December 2004, pp. 153–158.

[19] S. L. Smith, J. Tůmová, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," The International Journal of Robotics Research, vol. 30, no. 14, pp. 1695–1708, 2011.

[20] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, "Automated composition of motion primitives for multi-robot systems from safe LTL specifications," in 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2014, pp. 1525–1532.

[21] Y. Kantaros and M. M. Zavlanos, "Sampling-based optimal control synthesis for multirobot systems under global temporal tasks," IEEE Transactions on Automatic Control, vol. 64, no. 5, pp. 1916–1931, 2018.

[22] X. Luo and M. Zavlanos, "Transfer planning for temporal logic tasks," in Proc. of the 58th IEEE Conference on Decision and Control, France, Nice, 2019.

[23] M. Kloetzer, X. C. Ding, and C. Belta, "Multi-robot deployment from LTL specifications with reduced communication," in 2011 50th IEEE Conference on Decision and Control and European Control Conference. IEEE, 2011, pp. 4867–4872.

[24] Y. Shoukry, P. Nuzzo, A. Balkan, I. Saha, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada, "Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming," in 2017 IEEE 56th Annual Conference on Decision and Control (CDC). IEEE, 2017, pp. 1132–1137.

[25] S. Moarref and H. Kress-Gazit, "Decentralized control of robotic swarms from high-level temporal logic specifications," in 2017 International Symposium on Multi-robot and Multi-agent Systems (MRS). IEEE, 2017, pp. 17–23.

[26] B. Lacerda and P. U. Lima, "Petri net based multi-robot task coordination from temporal logic specifications," Robotics and Autonomous Systems, vol. 122, p. 103289, 2019.

[27] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems," The International Journal of Robotics Research, vol. 37, no. 7, pp. 818–838, 2018.

[28] Y. E. Sahin, P. Nilsson, and N. Ozay, "Multirobot coordination with counting temporal logics," IEEE Transactions on Robotics, 2019.

[29] K. Leahy, Z. Serlin, C.-I. Vasile, A. Schoer, A. M. Jones, R. Tron, and C. Belta, "Scalable and robust algorithms for task-based coordination from high-level specifications (scratches)," IEEE Transactions on Robotics, vol. 38, no. 4, pp. 2516–2535, 2021.

[30] Z. Liu, M. Guo, and Z. Li, "Time minimization and online synchronization for multi-agent systems under collaborative temporal tasks," arXiv preprint arXiv:2208.07756, 2022.

[31] L. Li, Z. Chen, H. Wang, and Z. Kan, "Fast task allocation of heterogeneous robots with temporal logic and inter-task constraints," IEEE Robotics and Automation Letters, 2023.

[32] D. E. Wilkins, Practical planning: extending the classical AI planning paradigm. Elsevier, 2014.

[33] I. Georgievski and M. Aiello, "Htn planning: Overview, comparison, and beyond," Artificial Intelligence, vol. 222, pp. 124–156, 2015.

[34] M. Weser, D. Off, and J. Zhang, "Htn robot planning in partially observable dynamic environments," in 2010 IEEE International Conference on Robotics and Automation. IEEE, 2010, pp. 1505–1510.

[35] B. Hayes and B. Scassellati, "Autonomously constructing hierarchical task networks for planning and human-robot collaboration," in 2016 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2016, pp. 5469–5476.

[36] L. H. De Mello and A. C. Sanderson, "And/or graph representation of assembly plans," IEEE Transactions on robotics and automation, vol. 6, no. 2, pp. 188–199, 1990.

[37] M. Colledanchise and P. Ögren, Behavior trees in robotics and AI: An introduction. CRC Press, 2018.

[38] Y. Cheng, L. Sun, and M. Tomizuka, "Human-aware robot task planning based on a hierarchical task model," IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 1136–1143, 2021.

[39] S. Baier and S. A. McIlraith, "Htn planning with preferences," in 21st Int. Joint Conf. on Artificial Intelligence, 2009, pp. 1790–1797.

[40] S. Lin and P. Bercher, "On the expressive power of planning formalisms in conjunction with ltl," in Proceedings of the International Conference on Automated Planning and Scheduling, vol. 32, 2022, pp. 231–240.

[41] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," Formal Methods in System Design, vol. 19, no. 3, pp. 291–314, 2001.

[42] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification," in 1st Symposium in Logic in Computer Science (LICS). IEEE Computer Society, 1986.

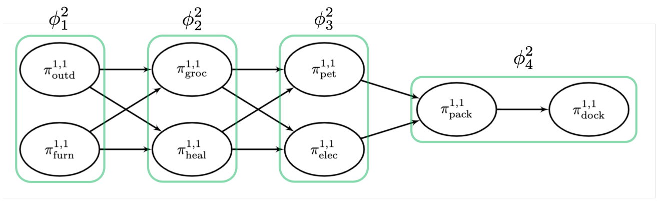
[43] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2018. [Online]. Available: http://www.gurobi.com

[44] P. Gastin and D. Oddoux, "Fast LTL to büchi automata translation," in International Conference on Computer Aided Verification. Springer, 2001, pp. 53–65.

APPENDIX I
FLAT FORM OF LTL TASKS

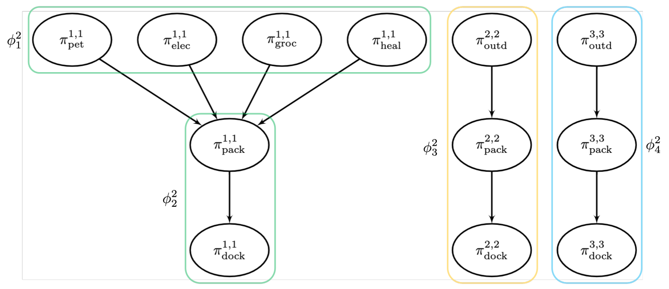
$$\boxed{\text{Task 1}}\quad \begin{aligned}\phi = {} & \Diamond(\pi^{1,1}_{\text{furn}} \wedge \bigcirc(\pi^{1,1}_{\text{furn}}\,\mathcal{U}\,\pi^{3,3}_{\text{furn}}))\\ & \wedge \Diamond(\pi^{3,3}_{\text{pack}} \wedge \Diamond\pi^{3,3}_{\text{dock}}) \wedge \neg\pi^{3,3}_{\text{pack}}\,\mathcal{U}\,\pi^{3,3}_{\text{furn}}\\ & \wedge \Diamond\pi^{1,1}_{\text{outd}} \wedge \Diamond\pi^{1,1}_{\text{pet}} \wedge \Diamond(\pi^{1,1}_{\text{pack}} \wedge \Diamond\pi^{1,1}_{\text{dock}})\\ & \wedge \neg\pi^{1,1}_{\text{outd}}\,\mathcal{U}\,\pi^{1,1}_{\text{furn}} \wedge \neg\pi^{1,1}_{\text{outd}}\,\mathcal{U}\,\pi^{3,3}_{\text{furn}}\\ & \wedge \neg\pi^{1,1}_{\text{pet}}\,\mathcal{U}\,\pi^{1,1}_{\text{furn}} \wedge \neg\pi^{1,1}_{\text{pet}}\,\mathcal{U}\,\pi^{3,3}_{\text{furn}}\\ & \wedge \neg\pi^{1,1}_{\text{pack}}\,\mathcal{U}\,\pi^{1,1}_{\text{outd}} \wedge \neg\pi^{1,1}_{\text{pack}}\,\mathcal{U}\,\pi^{1,1}_{\text{pet}}\\ & \wedge \Diamond\pi^{2,2}_{\text{heal}} \wedge \Diamond\pi^{2,2}_{\text{groc}} \wedge \Diamond(\pi^{2,2}_{\text{pack}} \wedge \Diamond\pi^{2,2}_{\text{dock}})\\ & \wedge \neg\pi^{2,2}_{\text{groc}}\,\mathcal{U}\,\pi^{2,2}_{\text{heal}} \wedge \neg\pi^{2,2}_{\text{pack}}\,\mathcal{U}\,\pi^{2,2}_{\text{groc}}\end{aligned}\tag{8}$$

$$\boxed{\text{Task 2}}\quad \begin{aligned}\phi = {} & \Diamond\pi^{1,1}_{\text{furn}} \wedge \Diamond\pi^{1,1}_{\text{outd}} \wedge \Diamond\pi^{1,1}_{\text{heal}} \wedge \Diamond\pi^{1,1}_{\text{groc}}\\ & \wedge \Diamond\pi^{1,1}_{\text{elec}} \wedge \Diamond\pi^{1,1}_{\text{pet}} \wedge \Diamond(\pi^{1,1}_{\text{pack}} \wedge \Diamond\pi^{1,1}_{\text{dock}})\\ & \wedge \neg\pi^{1,1}_{\text{heal}}\,\mathcal{U}\,\pi^{1,1}_{\text{furn}} \wedge \neg\pi^{1,1}_{\text{heal}}\,\mathcal{U}\,\pi^{1,1}_{\text{outd}}\\ & \wedge \neg\pi^{1,1}_{\text{groc}}\,\mathcal{U}\,\pi^{1,1}_{\text{furn}} \wedge \neg\pi^{1,1}_{\text{groc}}\,\mathcal{U}\,\pi^{1,1}_{\text{outd}}\\ & \wedge \neg\pi^{1,1}_{\text{elec}}\,\mathcal{U}\,\pi^{1,1}_{\text{heal}} \wedge \neg\pi^{1,1}_{\text{elec}}\,\mathcal{U}\,\pi^{1,1}_{\text{groc}}\\ & \wedge \neg\pi^{1,1}_{\text{pet}}\,\mathcal{U}\,\pi^{1,1}_{\text{heal}} \wedge \neg\pi^{1,1}_{\text{pet}}\,\mathcal{U}\,\pi^{1,1}_{\text{groc}}\\ & \wedge \neg\pi^{1,1}_{\text{pack}}\,\mathcal{U}\,\pi^{1,1}_{\text{elec}} \wedge \neg\pi^{1,1}_{\text{pack}}\,\mathcal{U}\,\pi^{1,1}_{\text{pet}}\end{aligned}\tag{9}$$



ϕ^2_1
ϕ^2_2
ϕ^2_3
ϕ^2_4
$\pi^{1,1}_{\text{outd}}$
$\pi^{1,1}_{\text{furn}}$
$\pi^{1,1}_{\text{groc}}$
$\pi^{1,1}_{\text{heal}}$
$\pi^{1,1}_{\text{pet}}$
$\pi^{1,1}_{\text{elec}}$
$\pi^{1,1}_{\text{pack}}$
$\pi^{1,1}_{\text{dock}}$

Fig. 5: Task network of task 2; see Equation (6).



ϕ^2_1
$\pi^{1,1}_{\text{pet}}$
$\pi^{1,1}_{\text{elec}}$
$\pi^{1,1}_{\text{groc}}$
$\pi^{1,1}_{\text{heal}}$
$\pi^{2,2}_{\text{outd}}$
$\pi^{3,3}_{\text{outd}}$
$\pi^{1,1}_{\text{pack}}$
ϕ^2_3
$\pi^{2,2}_{\text{pack}}$
$\pi^{3,3}_{\text{pack}}$
ϕ^2_4
ϕ^2_2
$\pi^{1,1}_{\text{dock}}$
$\pi^{2,2}_{\text{dock}}$
$\pi^{3,3}_{\text{dock}}$

Fig. 6: Task network of task 3; see Equation (7).

$$\boxed{\text{Task 3}}\quad \begin{aligned}\phi = {} & \Diamond\pi^{1,1}_{\text{heal}} \wedge \Diamond\pi^{1,1}_{\text{groc}} \wedge \Diamond\pi^{1,1}_{\text{elec}} \wedge \Diamond\pi^{1,1}_{\text{pet}}\\ & \wedge \Diamond(\pi^{1,1}_{\text{pack}} \wedge \Diamond\pi^{1,1}_{\text{dock}})\\ & \wedge \neg\pi^{1,1}_{\text{pack}}\,\mathcal{U}\,\pi^{1,1}_{\text{heal}} \wedge \neg\pi^{1,1}_{\text{pack}}\,\mathcal{U}\,\pi^{1,1}_{\text{groc}}\\ & \wedge \neg\pi^{1,1}_{\text{pack}}\,\mathcal{U}\,\pi^{1,1}_{\text{elec}} \wedge \neg\pi^{1,1}_{\text{pack}}\,\mathcal{U}\,\pi^{1,1}_{\text{pet}}\\ & \wedge \left(\Diamond\left(\pi^{2,2}_{\text{outd}} \wedge \Diamond\left(\pi^{2,2}_{\text{pack}} \wedge \Diamond\pi^{2,2}_{\text{dock}}\right)\right)\right.\\ & \left.\vee \Diamond\left(\pi^{3,3}_{\text{outd}} \wedge \Diamond\left(\pi^{3,3}_{\text{pack}} \wedge \Diamond\pi^{3,3}_{\text{dock}}\right)\right)\right)\end{aligned}\tag{10}$$