Autonomous Robots (2022) 46:667–683
https://doi.org/10.1007/s10514-022-10043-y

Hierarchical planning with state abstractions for temporal task specifications

Yoonseon Oh[1] · Roma Patel[2] · Thao Nguyen[2] · Baichuan Huang[3] · Matthew Berg[2] · Ellie Pavlick[2] · Stefanie Tellex[2]

Received: 6 May 2021 / Accepted: 17 May 2022 / Published online: 4 June 2022

Abstract
We often specify tasks for a robot using temporal language that can include different levels of abstraction. For example, the command *"go to the kitchen before going to the second floor"* contains spatial abstraction, given that "floor" consists of individual rooms that can also be referred to in isolation ("kitchen", for example). There is also a temporal ordering of events, defined by the word "before". Previous works have used syntactically co-safe Linear Temporal Logic (sc-LTL) to interpret temporal language (such as "before"), and Abstract Markov Decision Processes (AMDPs) to interpret hierarchical abstractions (such as "kitchen" and "second floor"), separately. To handle both types of commands at once, we introduce the Abstract Product Markov Decision Process (AP-MDP), a novel approach capable of representing non-Markovian reward functions at different levels of abstractions. The AP-MDP framework translates LTL into its corresponding automata, creates a product Markov Decision Process (MDP) of the LTL specification and the environment MDP, and decomposes the problem into subproblems to enable efficient planning with abstractions. AP-MDP performs faster than a non-hierarchical method of solving LTL problems in over 95% of path planning tasks, and this number only increases as the size of the environment domain increases. In a cleanup world domain, AP-MDP performs faster in over 98% of tasks. We also present a neural sequence-to-sequence model trained to translate language commands into LTL expression, and a new corpus of non-Markovian language commands spanning different levels of abstraction. We test our framework with the collected language commands on two drones, demonstrating that our approach enables robots to efficiently solve temporal commands at different levels of abstraction in both indoor and outdoor environments.

Keywords Hierarchical task planning · Linear temporal logic · Language

1 Introduction

In an ideal human-robot interaction scenario, humans would give robots tasks in the form of natural language utterances and gestures. The variation in language used allows for specifying tasks at varying levels of spatial abstractions, while specifying temporal constraints. Meaning can be conveyed with language at different levels of spatial abstraction, in terms of high-level goals (such as *"fly to the end of the first floor"*), lower-level specifications (such as *"fly east, go south, go south and fly east again"*), or mixed-level (such as *"go to the yellow room and the second floor"*). Language can also express explicit constraints on the path taken to reach the

✉ Yoonseon Oh
yoh21@hanyang.ac.kr

Roma Patel
romapatel@brown.edu

Thao Nguyen
thao_nguyen3@brown.edu

Baichuan Huang
baichuan.huang@rutgers.edu

Matthew Berg
matthew_berg@brown.edu

Ellie Pavlick
ellie_pavlick@brown.edu

Stefanie Tellex
stefie10@cs.brown.edu

[1] Department of Electronic Engineering, Hanyang University, Seoul 04763, Republic of Korea

[2] Department of Computer Science, Brown University, 115 Waterman Street, Providence, RI 02912, USA

[3] Department of Computer Science, Rutgers University, 110 Frelinghuysen Rd, Piscataway, NJ 08854, USA

goal (for example, *"fly to the red room first, without going through the green room."*). The former category of commands requires an agent to fluidly move within an abstraction hierarchy (that is, knowing that a *floor* is at a higher level than individual rooms and directions), while the latter command restricts the space of possible paths that can be taken and sometimes induces temporal constraints on the order in which goals can be visited. It is crucial for robot systems to portray an adequate understanding of such commands, coupled with the ability to efficiently execute the underlying task.

Given an environment, a goal condition and constraints, robots can use planning to reach goal conditions while satisfying constraints. Existing approaches interpret language by mapping to a reward function in a Markov Decision Process (MDP) (MacGlashan et al., 2015b). However, these models very quickly become intractable as the state space of the world grows larger (Gopalan et al., 2017; Konidaris, 2016). Planning with *abstractions* in a hierarchical structure (Gopalan et al., 2017; Konidaris, 2016; Konidaris et al., 2018; Sutton et al., 1999), either by using an Abstract Markov Decision Process (AMDP) (Gopalan et al., 2017) or with options (Konidaris et al., 2018; Konidaris, 2016; Sutton et al., 1999) can allow reduction of the state space. There has been previous work in interpreting natural language task specifications at different levels of spatial abstraction and planning using AMDPs (Arumugam et al., 2017). Separately, as shown in Fig. 1, non-Markovian natural language commands can be mapped to linear temporal logic (LTL) formulae (Boteanu et al., 2016; Finucane et al., 2010; Kress-Gazit et al., 2008; Lignos et al., 2015) to allow efficient planning with an MDP, given the corresponding LTL task specifications (Ding et al., 2011, 2014; Fu & Topcu, 2014; Gopalan et al., 2018; Kasenberg & Scheutz, 2017; Sadigh et al., 2014; Wolff et al., 2012). Combining the interpretation of language using a hierarchical structure and the mapping of commands to LTL expressions is non-trivial, as the non-Markovian constraints might span different levels of abstraction. Plans in a more abstract state space could therefore lead to failure of constraints specified in a less abstract space (that is, plans at a lower level in the abstraction hierarchy).

In our previous work (Oh et al., 2019), we have introduced the *Abstract Product MDP (AP-MDP)* framework to combine the benefits of LTL and AMDP, thus enabling a robot to interpret non-Markovian commands at different levels of abstraction. There is a hierarchical approach in planning for LTL tasks using options (Liu & Fu, 2018). However, the AMDP approach suits our task better, as its hierarchical structure closely resembles the hierarchies formed by humans when planning to solve complex tasks that can be decomposed into subtasks (Gopalan et al., 2017). In our approach, task specifications are first given as natural language utterances that are then translated into LTL expressions by a supervised neural sequence-to-sequence model. This LTL expression ϕ is converted into a finite state representation that accepts infinite inputs, or a Büchi automaton (Büchi, 1990). Since our planning problem considers only finite sequence of actions, we limit the translated LTL to syntactically co-safe LTL (sc-LTL). This representation allows us to decompose the problem into sub-problems (organized around sub-parts of the input LTL expression). Edges of the Büchi automaton consist of atomic propositions in expression ϕ and a sub-problem induces a state transition of the automaton. To further deal with different levels of abstraction, if atomic propositions in the same edge are from different levels, we solve the sub-problem using the lower level AMDP. The robot must then forgo the computational benefits of the AMDP to guarantee that the policy satisfies all the constraints present in the LTL expression. This entire pipeline (shown in Fig. 2) therefore fluidly allows complex task specifications with non-Markovian constraints to be specified using natural language and solved at different levels of the goal hierarchy.

In this paper, we provides in-depth evaluation of AP-MDP by applying the method in several domains. We evaluate our approach by reporting the performance of AP-MDP in

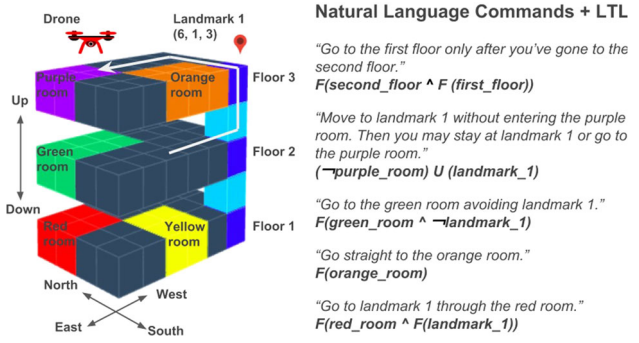


Fig. 1 Our environment is a gridworld with three floors, each consisting of rooms that consist of grid cells. The white arrow shows an example path the drone can take in the environment. We also include sample natural language commands (and their LTL formulae) that the drone successfully executed

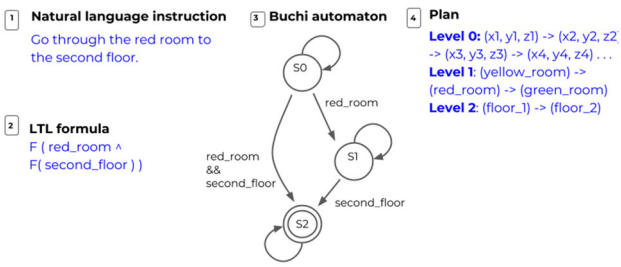


Fig. 2 Complete pipeline for the translation of a natural language instruction to an LTL formula, then to a Büchi automaton, and to a plan that gives us action sequences to correctly reach the goal location specified by the task

two simulation domains and on two drone platforms, one indoor and one outdoor. We also present a new corpus of non-Markovian natural language commands at different levels of abstraction, a neural sequence-to-sequence model that translates human-uttered natural language commands to their corresponding LTL counterparts, and demonstrates the solving of complex natural language task specifications using AP-MDP on two drones.

2 Related work

LTL has been used to model agent behavior in planning problems with non-Markovian task specifications. Consider a task that requires an agent to visit regions of interest in a specific order (for example, *"visit the red room first, then the blue room, and the green room last"*). These kinds of expressions have intrinsic temporal information that must be taken into account when determining the kind of path that has to be taken to achieve the goal. LTL allows us to formally describe these kinds of task specifications as logical functions, thus allowing robots to then execute these behaviors.

When the goal for a task is defined as an LTL expression, previous works have often formulated the problem as a product of an MDP and an automaton of the LTL formula (Ding et al., 2011, 2014; Fu & Topcu, 2014; Sadigh et al., 2014; Wolff et al., 2012). Some previous works model dynamic systems of agents as MDPs and developed methods to generate a control policy that satisfies LTL constraints (Ding et al., 2014, 2011). The LTL formula is converted into a *Deterministic Rabin Automaton* (DRA), and the dynamic system is formulated as a product of a DRA and MDP. The goal is then to search for a policy that satisfies the acceptance condition. Along the same lines, Kasenberg and Scheutz (2017) show that the reverse is also true, that is, the product of a DRA and an MDP can be considered to infer an LTL specification from demonstrations. However, this approach does not scale well for large MDPs.

Decision making with an MDP often becomes intractable as the size of the state space increases. In order to overcome intractability, hierarchical frameworks (Gopalan et al., 2017; Konidaris, 2016; Konidaris et al., 2018; Kulkarni et al., 2016) are commonly used. The options framework (Konidaris, 2016; Konidaris et al., 2018), for example, models temporally abstract macro-actions as options that can be adopted to build abstraction hierarchies. Similarly, AMDPs (Gopalan et al., 2017) can be used for abstraction by decomposing tasks into series of subtasks, thus allowing planning to take place more efficiently. However, these methods do not address the problem of solving LTL specifications with abstractions.

Hierarchical frameworks are powerful when an agent is faced with the task of planning a sequence of actions for complex LTL tasks. Several works (Cho et al., 2017; Fainekos et al., 2009; McMahon & Plaku, 2014; Oh et al., 2017) propose incorporating both the robot dynamics and the given LTL constraints in a continuous space. A continuous state space can be abstracted into a discrete state space and a continuous path is derived by sampling guided by the high-level discrete plan (Cho et al., 2017; McMahon & Plaku, 2014; Oh et al., 2017). Other works have focused on grounding natural language to LTL expressions (Boteanu et al., 2016; Gopalan et al., 2018; Lignos et al., 2015) to further allow a robot to make use of these LTL specifications. Previous work in hierarchical planning using options can accelerate planning for LTL tasks (Liu & Fu, 2018). However, the AMDP framework (Gopalan et al., 2017) is better suited for our task than options, by virtue of encoding a goal hierarchy rather than learning a policy over goals.

To the best of our knowledge, this work is the first to propose a hierarchical framework for planning for LTL tasks using the structure of an AMDP. An AMDP provides abstract states, actions, and transition dynamics in multiple layers above a base-level MDP, thus decomposing problems into subtasks with local rewards and local transition functions for policy generation. Moreover, as shown in our robot demonstrations, we start from human input given in the form of speech that is then converted to text. This textual input of the natural language command is translated to its LTL representation, and atomic propositions are directly mapped into propositions in each layer of a multi-level AMDP. We can then plan at levels higher than the lowest level whenever possible, and find a policy in a more efficient way than previous approaches.

3 Problem formulation

We consider a planning problem for a robot, when the task that the robot is required to interpret and solve is given through a natural language command. Our environment is a 3D grid world consisting of three floors as shown in Fig. 1. Each floor is composed of colored rooms, a room is composed of a set of grid cells, a landmark (such as a charging station) indicates a cell at position *(x,y,z)*. Landmarks (or cells) are therefore the lowest level of abstraction, rooms are abstract expressions of landmarks, and floors are abstract expressions of rooms and form the highest level of abstraction. A natural language command (such as *"first go to the red room through landmark 1 and then go to the blue room."*) is given to the robot by virtue of observable visual elements in our abstraction hierarchy (landmarks, rooms, and floors). This natural language utterance is grounded to its LTL counterpart ($\mathcal{F}$ (*landmark_1* $\wedge$ $\mathcal{F}$ (*red_room* $\wedge$ $\mathcal{F}$ (*blue_room*)))) which forms the task specification. The agent is required to accomplish the task by correctly finding a path to the correct location and following the determined path by executing a

sequence of actions from the action set (*north, south, east, west, up, down*).

We formulate this problem as an MDP that aims to accomplish the task as few actions as possible. Crucially, we make use of abstractions over the MDP state space for more efficient planning in large environments, and for the robot to efficiently find policies for commands at different levels of abstraction. Consider the example task above of *"first go to the red room through landmark 1 and then go to the blue room."* This is an expression that spans different levels in the abstraction hierarchy (that is, rooms and landmarks) and can be translated into its equivalent LTL formula ϕ over atomic proposition sets AP^L for each level L in the hierarchy. For example, *"landmark 1"* occupies one grid cell in the environment and corresponds to an atomic proposition (denoted by α_0^0) in AP^0 and *"red room"* and *"blue room"* correspond to atomic propositions (denoted by α_0^1 and α_1^1, respectively) in AP^1. The expression can be translated into $\phi = \mathcal{F}(\alpha_0^0 \wedge \mathcal{F}(\alpha_0^1 \wedge \mathcal{F}\alpha_1^1))$ using the LTL operator $\mathcal{F}$ or "finally", converted to a Büchi automaton (Büchi, 1990; Finucane et al., 2010), and then an AMDP (Gopalan et al., 2017) to decompose the problem into a series of smaller, and hence easier to solve, subproblems. Sect. 4 defines LTL and the variants of MDPs that our model relies on, while Sect. 5 goes over how they are composed together to produce a more efficient solution, while describing the end-to-end pipeline with the natural language grounding components.

4 Preliminaries

This section defines the components used in our formulation and how they are transformed into one another to form state abstractions for complex, non-Markovian task specifications uttered by humans through natural language. We briefly introduce LTL, syntactically co-safe LTL, and its syntax, explain the transformation of an LTL expression to a Büchi automaton and further to an MDP.

4.1 Linear temporal logic

Temporal logic was first introduced as a formalism for clarifying issues of time and defining the semantics of temporal expressions. LTL is a temporal logic whose syntax contains path formulae — the logical expression describes a specification that can be validated over a trajectory of any robot (discrete) system. LTL has the following grammatical syntax: $\phi ::= \pi \mid \neg\phi \mid \phi \wedge \varphi \mid \phi \vee \varphi \mid \mathcal{G}\phi \mid \mathcal{F}\phi \mid \mathcal{X}\phi \mid \phi\, \mathcal{U}\varphi$, where ϕ is the task specification or path formula, ϕ and φ are both LTL formulae, $\pi \in \Pi$ is an atomic proposition, $\mathcal{F}$ denotes "finally", $\mathcal{G}$ denotes "globally" or "always", $\mathcal{U}$ denotes "until", and $\neg$, $\wedge$, $\vee$ denote logical "negation", "and" and "or".

4.2 Linear temporal logic to deterministic Büchi automaton

An LTL formula intuitively expresses properties over trajectories or traces (a sequence of sets of atomic propositions) in the environment. This can be translated into an equivalent nondeterministic Büchi automaton (NBA). By Manna and Pnueli (1990), LTL formula can be classified according to its properties into the classes Guarantee, Safety, Obligation, Persistence, or Recurrence. Among them, a formula in Recurrence class can be translated into a deterministic Büchi automaton (Duret-Lutz, 2022). All linear temporal logic formulae generated by our language model belongs to Recurrence class. For convenient computation the NBA is constructed to a deterministic Büchi automaton (Bhatia et al., 2010), where a deterministic automaton differs from the general notion of automata in that it accepts infinite traces represented by the input LTL formula (Büchi, 1990). This handling of infinite traces is specifically necessary in cases of complex non-Markovian task specifications that can map to potentially unbounded action sequences.

Definition 1 *(Deterministic Büchi automaton)* A deterministic Büchi automaton (DBA) is a tuple $\mathcal{B} = (Q, \Sigma, \delta, q_0, \mathbf{F})$ where Q is a finite set of states, Σ is the input alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the initial state, and $\mathbf{F}$ is the acceptance condition.

For the LTL formula ϕ, the input alphabet of the automaton $\mathcal{B}$ is $\Sigma = 2^{AP}$. A word w over an alphabet can be any infinite sequence of atomic propositions, and the *run* of the automaton on $w = a_0 a_1 \cdots$, with $a_i \in \Sigma$ is a sequence of states $\rho = q_0 q_1 \cdots$, for $q_i \in Q$, where q_0 is an initial state and $q_{i+1} = \delta(q_i, a_i)$. A word is accepted by the automaton if and only if its run ρ satisfies the relationship $\inf(\rho) \cap \mathbf{F} \neq \emptyset$, where $\inf(\rho)$ is the set of states that occur infinitely often in r.

While DBA consumes an infinite sequence, the planning problem considers a finite sequence. So we assume that our language commands are limited to syntactically co-safe LTL formula (sc-LTL). Any infinite sequence satisfying an sc-LTL formula has a finite prefix which satisfies the sc-LTL formula. It contains only $\mathcal{X}, \mathcal{F}, \mathcal{U}$ temporal operators in positive normal form (i.e. $\neg$ only appears in front of atomic propositions) (Kloetzer & Mahulea, 2016). After this, LTL indicates sc-LTL.

4.3 Labeled Markov decision processes

In order to combine an MDP with the LTL formula to make an expanded MDP, we need to annotate each state with propositions so that we can evaluate the LTL expression. A labeled MDP (Fu & Topcu, 2014) is essentially an MDP where transitions are annotated with labels. These labels are provided

by a labeling function that maps states to valid propositions for each state.

Definition 2 *(Labeled MDP)* A labeled MDP is a tuple $\mathcal{M} = (S, A, T, s_0, AP, L, R)$, where S and A are finite state and action sets, $T : S \times A \times S \to [0, 1]$ is a transition probability function, $s_0 \in S$ is the initial state, AP is a set of atomic propositions, $L : S \to 2^{AP}$ is a labeling function which maps a state $s \in S$ into a set of atomic propositions valid at state s, and $R : S \to \mathbb{R}$ is a reward function.

4.4 Product Markov decision processes

We now need to combine the labeled MDP $\mathcal{M}$ with the LTL expression in order to make an expanded MDP which keeps track of the relevant parts of the LTL state. A product automaton is one that derives from the product of the finite transition system of $\mathcal{M}$ and the automaton $\mathcal{B}$ that represents the LTL specification. Labeled MDPs have previously been used for planning over an MDP to satisfy an LTL formula (Sadigh et al., 2014; Wolff et al., 2012), where the states of $\mathcal{M}$ and $\mathcal{B}$ encode the desired LTL specification. We can therefore design a state based reward function that relies on acceptance conditions of $\mathcal{B}$.

Definition 3 *(Product MDP)* Given a deterministic Büchi automaton $\mathcal{B} = (Q, \Sigma, \delta, q_0, \mathbf{F})$ and a labeled finite MDP $\mathcal{M} = (S, A, T, s_0, AP, L, R)$, with $s \in S$ and $q \in Q$, the product MDP (P-MDP) for the state (s, q) is given by $\mathcal{M}_p = (S_p, A, T_p, s_{0p}, Q, L_p)$ where:

(a) $S_p = S \times Q$ is a product state,
(b) $T_p((s, q), a, (s', q')) = \begin{cases} T(s, a, s'), & \text{if } q' = \delta(q, L(s')) \\ 0, & \text{otherwise,} \end{cases}$
(c) $s_{0p} = (s_0, q)$ such that $q = \delta(q_0, L(s_0))$,
(d) $L_p((s, q)) = q$,

4.5 Abstract Markov decision processes

An Abstract Markov Decision Process (Gopalan et al., 2017) (AMDP) hierarchy decomposes large planning problems into a series of subproblems with local reward and transition functions using state and action abstraction.

Definition 4 *(Abstract MDP)* An AMDP is a 6-tuple $\tilde{\mathcal{M}} = (\tilde{S}, \tilde{A}, \tilde{T}, \tilde{R}, \tilde{E}, F)$. These are the usual MDP components, with the addition of $F : S \to \tilde{S}$, a state projection function to map states from the original environment MDP into the AMDP abstract state space $\tilde{S}$. Actions in the action set $\tilde{A}$ of the AMDP are either primitive actions, or are associated with subgoals to solve in the environment MDP. The transition function $\tilde{T}$ captures the dynamics of the effects of changes in the AMDP state space once subgoals are completed. $\tilde{R}$ is the reward function. $\tilde{E} \subset \tilde{S}$ is the set of terminal states.

AMDP guarantees the recursive optimality at each level of abstraction. If the local reward and transition functions are correct, the whole policy is recursively optimal by the definition in Dietterich (2000). Though the local reward and value functions are not accurate, the error in its value and Q-value can be bounded as proved in Gopalan et al. (2017).

5 Technical approach

At a high level, we use a neural sequence-to-sequence model to convert an English command to the corresponding LTL expression, which is then translated to a Büchi automaton and then levels of the component AMDP to enable the robot to infer a policy based on the expression. We run a simulation that shows the produced action sequence, executable by a drone in a 3D environment.

5.1 Abstract labeled Markov decision processes

We propose *Abstract Labeled MDPs (AL-MDPs)* that decomposes an MDP $\mathcal{M}$ into multiple abstract labeled MDPs which are based on abstract states, actions, and transitions in multiple layers. The labeled MDPs in the lowest level, the ith level, and the highest level are denoted by $\hat{\mathcal{M}}^0$, $\hat{\mathcal{M}}^i$, and $\hat{\mathcal{M}}^L$, respectively. The abstract labeled MDP $\hat{\mathcal{M}}^i$ is defined below:

Definition 5 *(Abstract Labeled MDP):* $\hat{\mathcal{M}}^i = (\hat{S}^i, \hat{A}^i, \hat{T}^i, \hat{s}^i_0, AP, \mathcal{L}^i, R^i)$, where $\hat{S}^i$, $\hat{A}^i$, $\hat{T}^i$ and R^i are a set of states, a set of actions, a transition function, and a reward function, respectively. States in $\hat{\mathcal{M}}^i$ correspond to a combination of atomic propositions in AP by the labeling functions $\mathcal{L}^i : \hat{S}^i \to 2^{AP}$. The set of atomic propositions AP is a union of L disjoint sets AP^is, where $AP^i = \{\alpha^i_0, \cdots, \alpha^i_n\}$ (that is, $AP = \cup_{i=1}^{L} AP^i$). The proposition $\alpha \in AP$ belongs to AP^i, where i is the largest value which satisfies that there exists a state $s \in \hat{S}^i$ which can determine the truth value of α.

The abstract states, actions, atomic propositions are manually designed considering the problem domain. For example in a drone domain (Fig. 1), we have defined abstract states and actions as described in Sect. 3. Atomic propositions and labeling functions at each level are defined so that they correspond to an abstract state at the level where they exist. We define APs at the lowest level like `RobotAt(Landmark)`, because the unit of the lowest-level states is a grid. APs at the second level are `RobotIn(Room)`, where `Room` can be replaced with `Yellow_Room, Red_Room`, and so on, since the abstract states are represented by rooms at the second level. Similarly, at the highest level, APs at the highest level are `RobotOn(Floor)`. A labeling function corresponding to an AP will check if APs are true based on the

physical information of landmarks, rooms and, floors, where the robot exists.

5.2 Abstract product Markov decision processes

We propose *Abstract Product MDPs* (AP-MDPs) which combine AL-MDPs and DBAs to solve ordinary product MDPs efficiently. We furthermore show how our approach handles a combination of atomic propositions in multiple levels. For example, if some of the atomic propositions are defined at level 0, we cannot guarantee that a plan derived at level 1 or level 2 will satisfy level 0 constraints. This would require working at the lowest level of atomic propositions, thus losing the computational benefit of abstraction and a reduced state space. In all previous hierarchical approaches in this area, when atomic propositions of different levels exist together, the product MDP must be solved at the lowest level (level 0 in this case) to guarantee the satisfaction of the transition constraint that directly affects it. This therefore does not afford the computational benefit of planning at higher levels using AL-MDPs. Our approach, however, employs different depths of AL-MDPs by decomposing the product MDP into subproblems to benefit from the hierarchical structure when the LTL task includes atomic propositions at the lowest level.

AP-MDPs combine the automaton $\mathcal{B}$ of the LTL task specification with AL-MDPs. This involves taking an LTL formula in the form of an automaton, converting it to a labeled MDP and decomposing this MDP into several subproblems, each of which are individually solved at the required level of abstraction. We use a running example, as shown in Sect. 5.3 to highlight the process of how decomposed subproblems are solved for the task specification in question. Sect. 5.4 then explains how any problem can be decomposed into component subproblems and Sect. 5.5 presents the pseudocode for the algorithm for this process. The language grounding component of the system is discussed in Sect. 5.6.

5.3 Example problem

Consider the example in Fig. 3. This figure shows the DBA for the LTL task specification $\phi = \mathcal{F}(\alpha_0^0 \wedge \mathcal{F}\alpha_0^1)$ and we can see that the atomic proposition α_0^0 is in level 0 of the abstraction hierarchy, while α_0^1 is in level 1. To deal with these different levels in the abstraction hierarchy, we decompose the entire problem into different subproblems. The first subproblem $\hat{\mathcal{M}}_0$ is defined by a tuple $\hat{\mathcal{M}}_0 = (\hat{S}^0, \hat{A}^0, \hat{T}^0, \hat{s}_0^0, AP, \mathcal{L}^0, R^0)$ and here the agent wants to go to q_1 while not visiting other states in the DBA. The condition to reach the desired state, $f(q_0, q_1, s, s') = true$ is its *goal condition* and the condition to stay in the current state, $f(q_0, q_0, s, s') = true$ is its *stay condition*, where s and s' are the current state and the next state, respectively. The function f returns $true$ or $false$ depending on whether

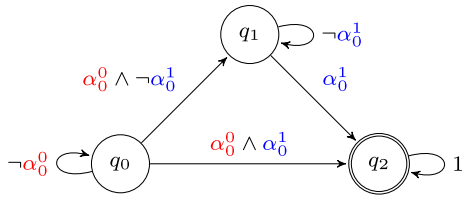


Fig. 3 Deterministic Büchi automaton. The transitions of the automaton refer to constraints over the propositions that are satisfied on taking that path. Red symbols and blue symbols represent atomic propositions in level 0 and in level 1, respectively

the logical expression on the edge is satisfied by the state. The reward function ensures that the agent gets a large positive reward if the goal condition is satisfied and gets a large negative reward if the stay condition is violated and the goal condition is not satisfied. In all other cases, it gets a small negative reward as the time taken increases. Since this subproblem contains atomic propositions at level 0, we can solve it at level 0, that is, the lowest level of atomic propositions.

We now consider the latter part of the decomposition, that is, the second subproblem $\hat{\mathcal{M}}_1$ from q_1 to q_2. This has atomic propositions related to level 1, therefore $\hat{\mathcal{M}}_1$ can be formulated at a higher level of abstraction (i.e. $\hat{\mathcal{M}}_1 = (\hat{S}^1, \hat{A}^1, \hat{T}^1, \hat{s}_0^1, AP, \mathcal{L}^1, R^1)$), allowing for more efficient planning over a smaller state space. In this way, all subproblems $\hat{\mathcal{M}}_i$ can be solved at the desired level to allow for full use of the benefits of abstraction where possible.

5.4 Subproblem decomposition

In this section, we describe how to decompose the Büchi automaton and construct AL-MDPs corresponding to subproblems. Let a run from the initial state q_0 to the accepting state q_a be $\rho = q_0^{n_0} q_1^{n_1} \cdots q_a$, where $q_i^{n_i}$ means that the state q_i is repeatedly visited n_i times. Since the automation does not reflect the physical configurations of environments, we extract subgoals from the automaton, construct subproblems to reach a subgoal in the form of AL-MDPs, and search for the plan by solving them. We call the sequence of subgoals a path ζ and a set of possible paths is a finite subset of accepted runs.

We generate paths based on the following criteria: (a) the path has a finite length, (b) the final state of the path should be an accepting state, and (c) the path does not visit the same state more than once. Since the robot stops as soon as it reaches the accepting state, the length of the successful plan is finite and the length of its corresponding run is finite, too. So the criteria (a) and (b) are valid. We add one more condition (c) to improve the efficiency of the plan. In these conditions, we can find a finite number of paths used to construct the subproblem. We assume that there are n_ζ paths and the ith path is denoted by $\zeta_i = q_0 q_1 \cdots q_{n_i}$, where q_{n_i} is

the accepting state. AP-MDPs decompose the problem into n_ζ subproblems each denoted by $\mathcal{P}_{\zeta_i}$, which accomplish the LTL task while following the path ζ_i.

Each problem $\mathcal{P}_{\zeta_i}$ can be decomposed into n_i subproblems, each formulated by an AL-MDP. Each subproblem $\mathcal{P}^j_{\zeta_i}$ aims to change the DBA state of the agent from q_j to q_{j+1}. Its *goal condition* and *stay condition* of $\mathcal{P}^j_{\zeta_i}$ are $f(q_j, q_{j+1}, s, s') = true$ and $f(q_j, q_j, s, s') = true$, respectively. The reward function for the AL-MDP is defined by:

$$R_j = \begin{cases} \gamma_{goal}, & \text{if } f(q_j, q_{j+1}, s, s') = true, \\ \gamma_{stay}, & \text{else if } f(q_j, q_j, s, s') = true, \\ \gamma, & \text{otherwise,} \end{cases} \quad (1)$$

where $\gamma_{goal} \gg 1$, γ_{stay} is a small negative value, and $\gamma \ll 0$. The large positive reward γ_{goal} drives the agent to reach the goal. The small negative reward γ_{stay} reduces the number of steps to reach the goal.

In this way, AP-MDPs can consist of $(\sum_{i=1}^{n_\zeta} n_i)$ AL-MDPs. Let's denote the plan for $\mathcal{P}_{\zeta_i}$ as $(s_{seq}, a_{seq})^{\zeta_i}$, where s_{seq} is the state sequence and a_{seq} the is action sequence. The plan for the LTL task is the shortest sequence $(s_{seq}, a_{seq})^*$ among all $(s_{seq}, a_{seq})^{\zeta_i}$s. For example, in Fig. 3, there are two paths, $\zeta_1 = q_0q_2$ and $\zeta_2 = q_0q_1q_2$ reaching the accepting state q_2. Let's assume that the solution of the subproblem $\mathcal{P}_{\zeta_1}$ corresponds to the run $q_0^{n_0}q_2$, where the robot stays q_0 for n_0 time steps. Solving a subproblem $\mathcal{P}_\zeta$ can be thought of as choosing the optimal one among an infinite number of runs considering physical environments.

5.5 Algorithm

The entire algorithm is presented as pseudocode in Algorithm 1. The input task is specified as an LTL expression composed of atomic propositions in the environment and the logical operators defined previously. We translate the LTL formula into a DBA using an existing package called Spot2 (line 4) (Duret-Lutz et al., 2016). Note that the DBA may contain infeasible edges because the translator does not consider the real environment (for example, if the *red_room* does not exist on the *first_floor* in a particular gridworld, `red_room`$\wedge$`floor_1` cannot be $true$). We handle this by eliminating edges which have contradictions consisting of a logical incompatibility between two or more propositions (line 5), based on specifications of the environment in question. We check the contradiction by looking at the truth table of the formula. Eliminating unrealizable edges improves the efficiency of the algorithm, because we do not have to solve AL-MDPs for which there is obviously no solution. We then find all possible paths from the initial state to the accepting state in line 6. The AL-MDPs *goal* and *stay* conditions are defined through lines 11 to 14, and we then obtain the optimal

Algorithm 1 Solve AP-MDPs

1: LTL task ϕ and s_0 are given
2: Initialize the optimal plan, $(s_{seq}, a_{seq})^*$.
3: Initialize the length of the optimal plan, l^*.
4: $A \leftarrow LTL2DBA(\phi)$
5: $A.RemoveContradiction()$
6: $Paths = A.FindPaths()$
7: **for** $\zeta_i \in Paths$ **do**
8: Initialize s_0
9: Initialize the plan $(s_{seq},\ a_{seq})^{\zeta_i}$
10: **for** j in $\{0, \cdots, n_i - 1\}$ **do**
11: *goal condition* $\leftarrow f(q_j, q_{j+1}, s, s') = true$
12: *stay condition* $\leftarrow f(q_j, q_j, s, s') = true$
13: $\ell_j \leftarrow$ the lowest level of atomic propositions in *goal* and *stay conditions*.
14: $\hat{\mathcal{M}}_j \leftarrow (\hat{S}^{\ell_j},\ \hat{A}^{\ell_j},\ \hat{T}^{\ell_j},\ \hat{s}_0^{\ell_j},\ AP,\ \mathcal{L}^{\ell_j},\ R^{\ell_j})$.
15: $ss, aa \leftarrow Solve(\hat{\mathcal{M}}_j)$
16: $(s_{seq},\ a_{seq})^{\zeta_i} \leftarrow (s_{seq},\ a_{seq})^{\zeta_i} \cup (ss, aa)$
17: $s_0 \leftarrow s_{seq}(end)$
18: **end for**
19: **if** $length(s_{seq}) < l^*$ **then**
20: $(s_{seq}, a_{seq})^* \leftarrow (s_{seq}, a_{seq})^{\zeta_i}$
21: **end if**
22: **end for**

Algorithm 2 $Solve(\mathcal{M}, a_{seq}, s_{seq})$

1: Input: $\mathcal{M}, a_{seq}, s_{seq}$
2: Output: s_{seq}, a_{seq},
3: $l \leftarrow$ the level of $\mathcal{M}$
4: **if** $\mathcal{M}$ is in the lowest level **then**
5: $ss, aa \leftarrow Plan(\mathcal{M})$
6: $s_{seq} \leftarrow [s_{seq}, ss[1 : end]]$
7: $a_{seq} \leftarrow [a_{seq}, aa[0 : end]]$
8: **else**
9: $ss, aa \leftarrow Plan(\mathcal{M})$
10: $s \leftarrow s_{seq}[0]$
11: **for** a in aa **do**
12: $\mathcal{M}' \leftarrow BuildLowerMDP(s, a, l - 1)$
13: $s_{seq}, a_{seq} \leftarrow Solve(\mathcal{M}', s_{seq}, a_{seq})$
14: $s \leftarrow s_{seq}[end]$
15: **end for**
16: **end if**

policy and plan of AL-MDPs with a variation of the solver of the AMDP (line 15). We then select the best plan which has the minimum number of actions (lines 19-21).

The algorithm $Solve$ is described in Algorithm 2. If the input MDP $\mathcal{M}$ is defined in the lowest level, we can solve the problem using the standard MDP solver (line 5). In this paper, we choose value iteration. Then the function returns the sequence of primitive actions and states. If the input MDP $\mathcal{M}$ is not defined in the lowest level, $Plan(\mathcal{M})$ returns sequences of non-primitive actions and states (line 9). In order to execute action a in the resulted action sequence aa, we need to define M' corresponding to the action a (line 12). Then we call $Solve()$ recursively, until it returns the lowest level of actions and states (line 13). The function $BuildLowerMDP$ grounds the action a with the level l into the lower level of MDP with the initial state s in the lowest level (Algorithm 3).

Algorithm 3 $BuildLowerMDP(a, s, l)$

1: Input: s, a, l
2: Output: MDP $\mathcal{M}$
3: $S \leftarrow$ the state space at the level l
4: $s_0 \leftarrow mapping\_state(s)$
5: $A \leftarrow define\_action\_set(s, a)$
6: $R \leftarrow define\_reward\_function(s, a)$

$$R = \begin{cases} \gamma_{goal}, & \text{if } f_{goal}(s) = true, \\ \gamma_{stay}, & \text{else if } f_{stay}(s) = true, \\ \gamma, & \text{otherwise,} \end{cases} \quad (2)$$

7: $\mathcal{M} \leftarrow (S, A, T, s_0, AP, L, R)$

The $mapping\_state$ function projects s to s_0, the state at the level $l-1$ (line 4). While AMDP (Gopalan et al., 2017) shares the same action space with all MDPs in the same level, we can dynamically choose the action space depending on the current action and state (line 5). Similarly, the reward function also dynamically defined depending on the current action and state (line 6). For example, let's ground the action 'Go to the first floor', when a robot is in the second floor. The goal condition $f_{goal} : S \mapsto \{true, false\}$ becomes true, when a robot is in the first floor. Additionally, we define the $stay$ condition f_{stay}, which becomes $true$ only if the robot is in the current floor (e.g. second floor). Then we can reduce the state space, because we can explore only first and the second floors. We will show the example of choosing the action space depending on the given action and state, in the Sect. 6.

If we have correct local reward and transition functions, we can guarantee the recursive optimality of the whole problem, though the solver only goes from a high-level to a lower level. We have decomposed AP-MDP into n_p subproblems, $P_{\zeta_i} s$. Each problem P_{ζ_i} is decomposed into n_i subproblems, formulated by AL-MDP in Algorithm 1. Since the strategy to solve AP-MDP is basically same as the abstract MDP, the optimality properties described in Sect. 4.5 hold for each subproblem. Then we search for the policy exhaustively with respect to all paths in Büchi automaton and choose the policy which takes the minimum number of actions. So the recursive optimality holds, if each AP-MDP satisfies the condition, the local reward and transition functions are correct.

5.6 Grounding language to LTL formulae

We train a neural sequence-to-sequence model to translate natural language commands to LTL expressions. We discuss our language corpus and the model architecture below.

5.6.1 Corpus

We use Amazon Mechanical Turk (AMT) to collect non-Markovian natural language commands that also refer to elements in the environment at different levels of abstraction.[1] AMT workers were shown images representing correct and incorrect ways for the robot to complete a task, and asked to give commands that accurately capture the robot's correct behavior. 810 natural language commands were collected from 120 AMT workers for 27 LTL formulae. We augment these 810 commands to obtain 6185 commands for 343 LTL expressions. Augmentation is done by mapping one training sample (for example, *"go to the red room"* accompanied by $\mathcal{F}(red\_room)$) to similar commands and corresponding LTL expressions for every other possible goal locations. We held aside 20% of the data as the test set to evaluate model performance and trained on all remaining data and perform 5 fold cross-validation in this manner.

5.6.2 Sequence-to-sequence model

As in Gopalan et al. (2018), we use a neural sequence-to-sequence model composed of a recurrent neural network (RNN) encoder and decoder to translate each natural language instruction to an LTL formula. It is implemented in PyTorch (Paszke et al., 2017) and trained for 10 epochs over our corpus, with a learning rate of 0.001 using the Adam optimizer (Kingma & Ba, 2014). We used a dropout of 0.8 as a regularizer (Srivastava et al., 2014).

6 Abstract Product MDP in Cleanup World

This section describes AP-MDP for a larger domain, Cleanup World (MacGlashan et al., 2015a), shown in Fig. 7a. In Cleanup World, a robot can pick and place objects, move objects to specific rooms, and move to a room or an object's location. The state space and action space grows combinatorially with the number of objects and rooms. We increase the planning efficiency by dynamically defining the action space depending on the goal.

6.1 Problem formulation

We consider an $n_x \times n_y$ grid world consisting of rooms with `RoomID` and objects with `ObjectID`. `RoomID` and `ObjectID` have the form `R#`*number* and `O#`*number*, respectively (e.g., `R0` represents the 0th room and `O0` represents the 0th object). In the level 0, a robot's state is defined as $[x_r, y_r, \texttt{ObjectID/NONE}]$, where $[x_r, y_r]$ is the robot's location in the grid world and `ObjectID/NONE` denotes the object the robot is carrying. An ith object's state is defined as its location $[x_i, y_i]$. The environment state for level 0 is defined as $X = [x_r, y_r, \texttt{ObjectID/NONE}, x_1, y_1, \cdots, x_n, y_n]$, where n is the number of objects in the grid world. The robot's action set includes {`north`,

[1] The corpus can be found at https://github.com/h2r/ltl-amdp

east, west, south, pickup, place}.
Each action north/east/west/south moves the robot one grid in the corresponding direction. The robot can pickup an object, if it is at the object's location. It can place the carrying object, if it not at the object's location and it does not break connectivity of adjacent rooms. For the level 1, the robot's state is defined as [RobotLocation, ObjectID/NONE], where RobotLocation is [ObjectID/NONE, RoomID/NONE]. For example, if a robot is at the same location as object Oi in room Rj and is not carrying Oi, RobotLocation $=$ [Oi, Rj]. The ith object's state is o_i $=$ RoomID, the room it is currently in, which is updated regardless of whether the object is being carried by the robot. The environment state for level 1 is Y $=$ [RobotLocation, ObjectID/NONE, $o_1, \cdots, o_n$]. Actions for level 1 include Nav(RoomID), Nav(ObjectID), PICKUP(Oi), and PLACE. A robot can move to a room with RoomID through Nav(RoomID), while avoiding objects. It can also move to the the location of an object with ObjectID through Nav(ObjectID), if the robot is currently in the room with that object. It can execute PICKUP(Oi), if the robot is at object Oi's location. When the robot executes PLACE, it places the object it is carrying in the room it is currently in. For level 2, we do not consider the robot's location. The environment state for level 2 is Z $=$ [ObjectID/NONE, $o_1, \cdots, o_n$], where the first term represents the object being carried by the robot. The robot actions for level 2 are Moveto(RoomID), Activate(ObjectID), and Deactivate. The action Activate(ObjectID) makes the robot move to the object ObjectID and pick it up. The action Deactivate makes the robot unload the object. The action Moveto (RoomID) makes the activated object to the adjacent room with RoomID.

The atomic proposition for LTL specifications at level 2 is IN(ObjectID, RoomID), which means that an object with ObjectID is in a room with RoomID, and the object is not on the robot. Atomic propositions for level 1 are ON(ObjectID) – which is true, if the object with ObjectID is on (being carried by) the robot, RobotIn (RoomID) – which is true, if a robot is in a room with RoomID, and RobotAt(ObjectID) – which is true, if a robot is at the location of an object with ObjectID.

The language command "Move the red object to the red room and then move the blue object to the blue room" can be translated to $\mathcal{F}$(IN(O0, R0) $\wedge$ $\mathcal{F}$IN(O1, R1)), where O0 and O1 are red and blue objects, and R0 and R1 are red and blue rooms, respectively. The command "Pick the red object up, go to the red room, and then place it" can be translated into $\mathcal{F}$(ON(O0) $\wedge$ $\mathcal{F}$(RobotIn(R0) $\wedge$ ON(NONE)). The atomic propositions do not directly correspond to the level 0 goals.

6.2 AP-MDP in Cleanup World

Though we employ the hierarchical structure for planning, MDP problem in each level is still too complex because of the large action space and state space. The function $define\_action\_set(s, a)$ and $define\_reward\_function(s, a)$ in Algorithm 3 reduce the size of the action space and the state space. The function, $define\_action\_set(s, a)$, will define the action space for Activate(O1) as a set of Nav(RoomID) with all RoomID, Nav(O1) and PICKUP(O1). The action Nav (O1) is grounded into the lowest level of MDP with the action space {north, east, west, south}, because we do not need pickup and place for navigation. The state space is limited to the room in which the is currently located by defining

$$f_{stay}(s) = \begin{cases} True, & \text{if } s \text{ is in the current room,} \\ False, & \text{otherwise.} \end{cases}$$

The command Nav(O1) can be executed, only if the robot is in the room, in which object O1 exists. Then we can prevent the action space and the state space from growing up exponentially, as the number of objects and rooms increases.

7 Simulations

In this section, we show that our method efficiently generates plans for complex LTL tasks. We successfully applied the proposed method on a drone, as well as a much larger cleanup domain. We evaluate efficiency with the number of backups and the computation time over 100 tasks for each domain.

7.1 Drone domain

7.1.1 Environment setup

For simulations, we consider two 3D grid worlds ($\mathcal{E}_1$ and $\mathcal{E}_2$) of size $6 \times 4 \times 3$ and $30 \times 20 \times 6$, respectively. The smaller world $\mathcal{E}_1$ has three floors, each comprised of six rooms, each the size of 2×2 grid cells. The larger world $\mathcal{E}_2$ has six floors, each comprised of six rooms of size 10×10. The visually observable elements (grid cells, rooms and floors) form the atomic propositions of the LTL task specifications. Importantly, these elements span different levels of abstraction: landmarks (grid cells) are at level 0, rooms are at level 1, and floors are at level 2. While our simulation environments consist of at least three floors, our robot demonstration is performed in a gridworld with only two floors for compatibility with the maximum height our PiDrone can reach.

7.1.2 Examples in simulation

We consider the tasks below to demonstrate example simulations of our proposed method. We show the language command with the corresponding LTL task specification, the automaton of the LTL expression, and the path found by our proposed approach for each example. This highlights how our method solves a given task while satisfying the constraints of the task. The tasks in question exhibit the complex constraints with non-Markovian nature and varying levels of abstraction as outlined above. They contain propositions at different levels in the abstraction hierarchy, and contain temporal order constraints by specifying certain subtasks that should be performed before others. The two tasks are:

1. ϕ_1=$\mathcal{F}$ ((`floor_2` $\vee$ `red_room`) $\wedge$ $\mathcal{F}$(`floor_1`))
   ("First either go to the second floor or the red room, and then go to the first floor")
2. ϕ_2 = $\mathcal{F}$(`floor_2` $\wedge$ $\mathcal{F}$(`green_room`))
   ("Go to the green room after entering the second floor")

The execution of both tasks is shown in Fig. 4. The process to solve task ϕ_1 for the given LTL task specification is outlined in the left side of the figure. Upon decomposing this task specification as in our proposed method, there are two paths of automaton states. Consider the path $\zeta_0 = q_0q_2$ corresponding to the AL-MDP $\hat{\mathcal{M}}_0$. This has a goal condition of ((`red_room` $\wedge$ `floor_1`) $\vee$ (`floor_2` $\wedge$ `floor_1`)) and a stay condition of ($\neg$`floor_2` $\wedge$ $\neg$`red_room`). For the path $\zeta_1 = q_0q_1q_2$, there are two AL-MDPs $\hat{\mathcal{M}}_0$ and $\hat{\mathcal{M}}_1$, where $\hat{\mathcal{M}}_0$ has a goal condition of ((`red_room` $\wedge$ $\neg$ `floor_1`) $\vee$ (`floor_2` $\wedge$ $\neg$`floor_1`)) and a stay condition of ($\neg$`floor_2` $\wedge$ $\neg$`red_room`), and $\hat{\mathcal{M}}_1$ has a goal condition of (`floor_1`) and a stay condition of ($\neg$`floor_1`). Since we can satisfy ϕ_1 with only two actions with ζ_0, the final solution is a plan for ζ_0.

For task ϕ_2, there exists an infeasible path among paths in the automaton. The first AL-MDP in $\zeta_0 = q_0q_2$ has goal and stay conditions of (`floor_2` $\wedge$ `green_room`) and ($\neg$`floor_2`), respectively. This problem does not have a solution because the green room is on the second floor, and thus our algorithm does not return a plan. There is, however, a solution for the path $\zeta_1 = q_0q_1q_2$. The first AL-MDP has a goal condition of (`floor_2` $\wedge$ $\neg$`green_room`) and a stay condition of ($\neg$`floor_2`). The second AL-MDP has a goal condition of (`green_room`) with a stay condition of ($\neg$`green_room`). The planned path is shown in Fig. 4.

7.1.3 Efficiency

In this section, we evaluate the efficiency of the proposed algorithm by measuring the computing time and the number of backups of the algorithm. The measured computing time includes pre-processing time like translating the LTL expression to a DBA and searching for a path in the DBA, along with the final planning time. The hierarchical structure allows for more efficient planning when unnecessary backup across multiple levels of the hierarchy is limited. We also evaluate the ability of different models to plan without this unnecessary computation. For each problem, the number of backups depends on the number and size of subproblems.

Since planning for an LTL task can be formulated as the product of an automaton $\mathcal{B}$ and MDP $\mathcal{M}$ as described in Sect. 4.4, our baseline algorithm (called P-MDP) is one that solves the product MDP at level 0 using value iteration. We ran 100 random tasks in the aforementioned environments ($\mathcal{E}_1$ and $\mathcal{E}_2$). The example tasks here are LTL specifications randomly sampled from the set

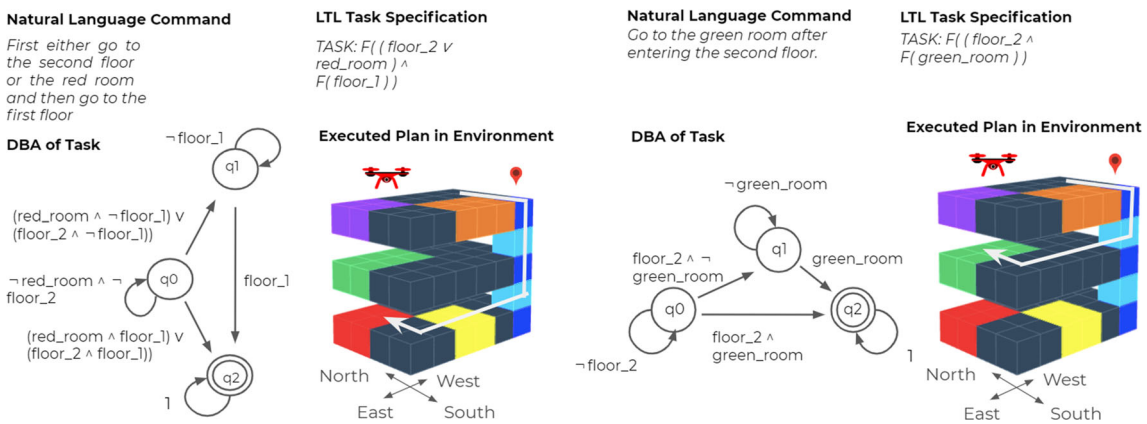


Fig. 4 Examples, left and right, tested in simulation. In each example, a natural language instruction is converted to an LTL expression, then to a corresponding AP-MDP to find a policy. An agent then executes the policy in the specified environment to reach the correct goal state through the desired path

$\{\mathcal{F}a,\ \mathcal{F}(a \wedge \mathcal{F}b),\ \mathcal{F}(a \wedge \mathcal{F}(b \wedge \mathcal{F}c)),\ \mathcal{F}a \wedge \mathcal{F}b,\ \neg a\ \mathcal{U}\ b\}$, where a, b, and c are atomic propositions that can be visually observed in our environment (such as `landmark_1`, `green_room`, `first_floor`). We ensure that atomic propositions are sampled from all possible landmarks, rooms, and floors to get a full variety of commands, and ensure that environment constraints are satisfied. For example, if level 1 is sampled, we sample the index of rooms among all possible rooms in that level. The lowest level of sampled atomic propositions is denoted by 0.

We display the results as histograms plotted in Figs. 5 and 6. In Fig. 5, the y-axis denotes the cumulative number of cases evaluated, while the x-axis denotes the computing time and the number of backups. We plot results for both environments $\mathcal{E}_1$ (on the left) and $\mathcal{E}_2$ (on the right). The red line shows computing time taken, while the blue line shows the number of backups, and the dotted line refers to the P-MDP (our baseline) while the bold line refers to the AP-MDP (our proposed model). For the corresponding number of cases on the y-axis, we can see the time taken or the number of backups, as plotted by the four lines. In both environments $\mathcal{E}_1$ and $\mathcal{E}_2$, the AP-MDP finds solutions with a shorter computing time and a smaller number of backups in the majority of cases. The size of environment $\mathcal{E}_2$ is much larger than $\mathcal{E}_1$, and it therefore takes longer computing time and more backups. It should be noted that AP-MDP perform significantly better than P-MDP given the benefits of abstraction in large states spaces.

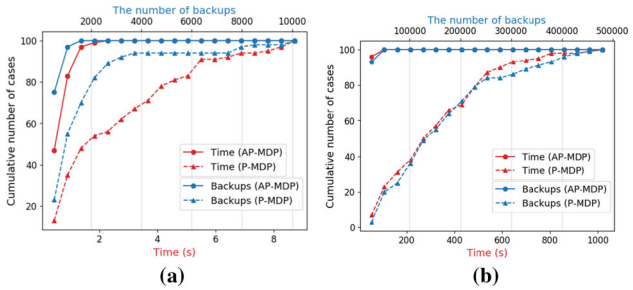


Fig. 5 Cumulative histograms of computing time and the number of backups of AP-MDP and P-MDP in the environment **a** $\mathcal{E}_1$ and **b** $\mathcal{E}_2$. We execute AP-MDP and P-MDP with 100 random LTL tasks in two environments, $\mathcal{E}_1$ and $\mathcal{E}_2$. The y-axis shows the cumulative number of cases evaluated

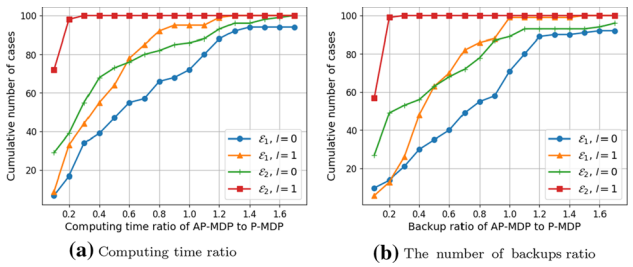


Fig. 6 Cumulative histograms of **a** computing time ratio (lower is better) and **b** the number of backups ratio (lower is better) of AP-MDP to P-MDP

In Fig. 6, to compare the efficiency of the two algorithms we plot the ratio (that is, AP-MDP to P-MDP) for the same metrics. For both computing time and number of backups, a ratio less than 1.0 indicates that AP-MDP is more efficient than P-MDP. The y-axis shows the cumulative number of cases, while the x-axis shows the ratio of the computing time taken. For a corresponding ratio on the x-axis ($r = 0.2$, for example) we can see the number of cases that had a ratio $< r$). Therefore, a line that solves a larger number of cases (out of 100) at a smaller ratio is a better solution. The four lines refer to different environments when solved at different levels. For example, ($\mathcal{E}_1$, $l = 1$) refers to the smaller environment at level 1. In $\mathcal{E}_1$, AP-MDP is better in 72 among the 100 cases with respect to the computing time and for 71 cases with respect to the number of backups. In $\mathcal{E}_2$, AP-MDP is better in 86 among 100 cases with respect to the computing time and for 89 cases with respect to the number of backups. The AP-MDP decomposes the problem and therefore has to solve more MDPs than the P-MDP. This means that in certain cases, especially in the smaller environment where abstraction is unnecessary, this approach is not faster. However, in the larger environment, employing abstraction increases the efficiency by reducing the size of each problem. To clearly show the effect of abstraction, we run simulations with atomic propositions in higher levels (AP^1 and AP^2), to assess how much abstraction helps when dealing with high-level commands. In $\mathcal{E}_1$, the computing time ratio is less than 1.0 in 95 cases and the number of backups ratio is less than 1.0 in 99 cases. In the larger environment $\mathcal{E}_2$, the computing time ratio and the number of backups ratio are less than 1.0 in all cases.

7.1.4 Language grounding results

We observe that while the model achieves 92.2% accuracy on a test set of commands that it saw during training, its accuracy drops to 68.3% on the held-out LTL commands. We observe that the accuracy of the model drops on the held-out LTL commands. This problem of zero-shot generalization (specifically, the ability to generalize to samples unseen during training) has been widely studied (Gopalan et al., 2018; Koehn & Knowles, 2017; Lake & Baroni, 2017) for neural sequence-to-sequence models that cannot handle compositionality and the ability of models to learn meaning representations for given natural language sentences (Dasgupta et al., 2018). We also observe cases where changes in word order affect the translated LTL output of the model. Consider the command *"avoid the blue room until you go to landmark 1"*, ($\neg$*blue_room* $\mathcal{U}$ *landmark_1*) for example. Variations in our collected data include sentences like *"until you go to landmark 1, always*

avoid the blue room" that change the ordering of referent words (*blue_room* and *landmark_1*) which are occasionally confused, and mapped to incorrect expressions such as ($\neg$*landmark_1* $\mathcal{U}$ *blue_room*). However, in the drone demonstrations, the sequence-to-sequence model correctly translate the given language commands (converted from speech) into LTL task specifications that are then solved using our proposed method.

7.2 Cleanup world domain

In this section, we successfully applied AP-MDP planning to more complex cleanup tasks.

7.2.1 Environment setup

We consider a 2D gird world of size 8×11 with five colored rooms as shown in 7. There are n_o colored objects and black walls. A robot avoids an object unless it picks it up while navigating the environment. The robot can place the object unless the object does not cut the connectivity of adjacent rooms. The robot can bring only one object at once.

7.2.2 Examples in simulation

Fig. 7 shows example scenarios for various tasks. A robot moves along the line. The color of the line is same as the color of the object which a robot is carrying. If the robot does not bring any object, the line is black. For Fig. 7(a)-7(e), the lowest level of atomic propositions is one. We plan in the level 1 first, and then plan in the level 0, for these tasks. In Fig.7(a), the plan at the level 1 of $\mathcal{F}$`RobotAt(O0)` is `Nav(R1), Nav(R0), Nav(O0)`. AP-MDP can handle various tasks like $\mathcal{F}$`RobotIn(R2)`, $\mathcal{F}$`RobotIn(R0)` $\wedge$ $\mathcal{F}$`RobotIn(R2)`, $\mathcal{F}$(`ON(O0)` $\wedge$ $\mathcal{F}$`RobotIn(R2)`), and $\neg$`RobotIn(R1)`$\mathcal{U}$`IN(O0, R4)` in Fig. 7(b)-7(e).

The lowest level of atomic propositions of Fig. 7(f)-7(h). The plan at level 2 for $\mathcal{F}$`IN(O1, R0)` $\wedge$ $\mathcal{F}$`IN(O0, R3)` is `Activate(O1), Moveto(R0), Deactivate, Activate(O0),Moveto(R2),Deactivate` (Fig. 7(f)). `Activate(O1)` corresponds to `Nav(R4), Nav(R1),`

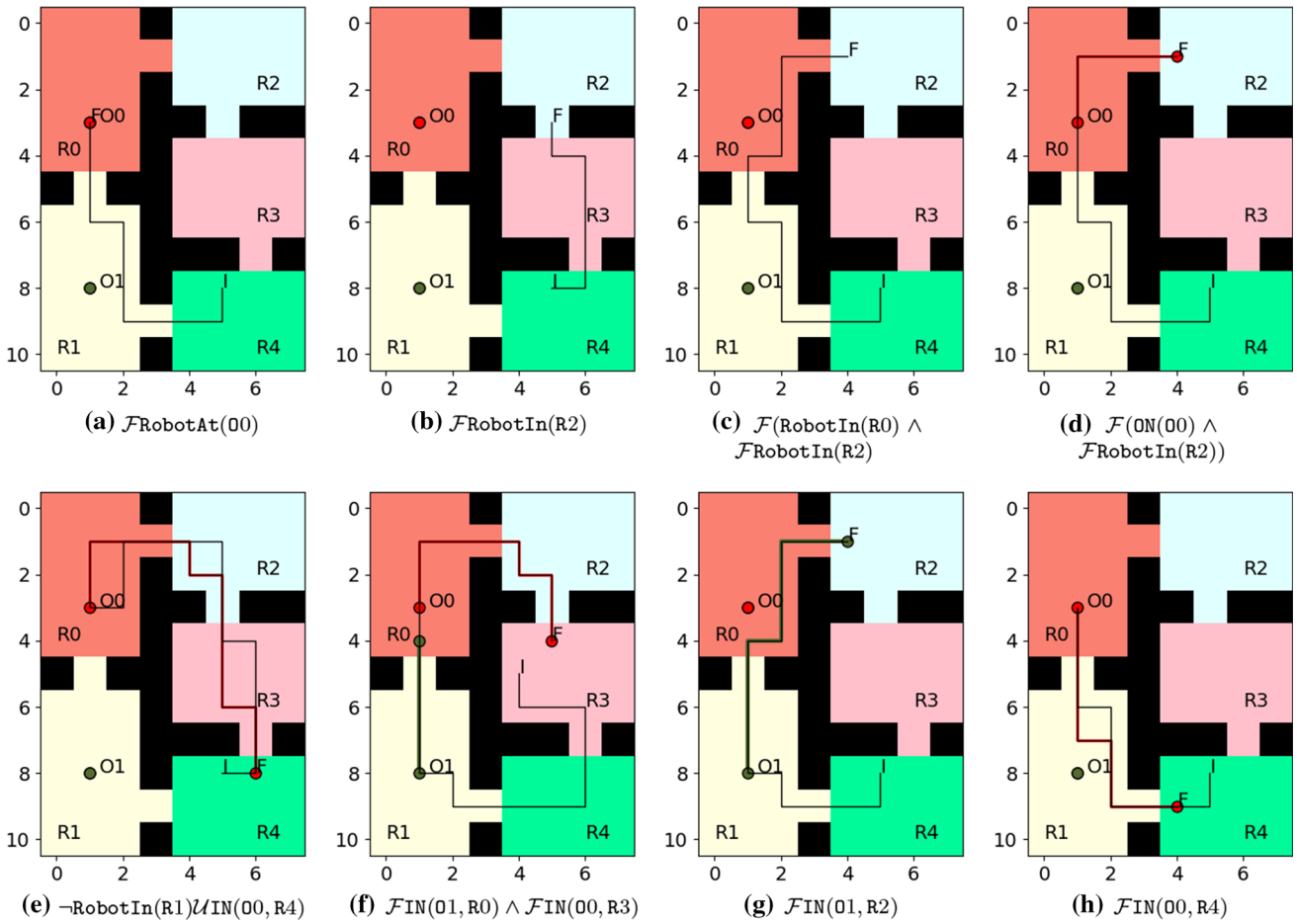


Fig. 7 Example scenarios in a Cleanup World. A robot starts at I, follows the black line, and stops at F. A colored line represents the trajectory of the object with that color (Color figure online)

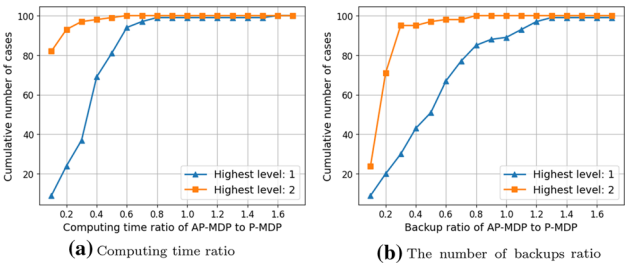
`Nav(O1)`, `PICKUP(O1)` at level 1. `Moveto(R0)` corresponds to `Nav(R0)` and `Deactivate` corresponds to `PLACE`. The plan at level 2 for $\mathcal{F}$`IN(O1, R2)` is `Activate (O1)`, `Moveto(R0)`, `Moveto (R2)`, `Deactivate` (Fig. 7(g)). Similarly, the plan at level 2 for $\mathcal{F}$`IN(O0, R4)` is `Activate(O0)`, `Moveto(R1)`, `Moveto(R4)`, `Deactivate` (Fig. 7(h)).

7.2.3 Efficiency

The efficiency of AP-MDP is evaluated by measuring the computing time and the number of backups compared with the baseline algorithm as describing in a drone domain. We ran 100 random tasks in the environment. The LTL specifications are randomly sampled from the set $\{\mathcal{F}a,\ \mathcal{F}(a \wedge \mathcal{F}b),\ Fa \wedge Fb,\ \neg a\mathcal{U}b\}$, where a and b are atomic propositions randomly sampled from possible atomic propositions (e.g. `RobotIn(R3)`, `In(O0,R4)`, `RobotAt(O3)`, `On(O0)`).

The results are shown as histograms plotted in Figs. 8 and 9. In Figs. 8, we plot the ratio, AP-MDP to P-MDP for the computing time and the number of backups. The number of objects is fixed to 2. AP-MDP is more efficient than P-MDP, if the ratio less than 1.0. The y-axis shows the cumulative number of cases, and the x-axis shows the ratio of the computing time taken and the number of backups. We compare performance for LTL tasks of which highest level of atomic propositions is 1 (a blue line) and 2 (an orange line). Though AP-MDP is better in most cases for both cases, the efficiency dramatically increases, if the LTL specification contains atomic propositions at the level 2. P-MDP takes much more time to solve the task with higher abstraction.

In Figs. 9, we show the number of backups and time with respect to one to four objects. Atomic propositions are sampled at the level 1 (i.e. we do not consider `In`), because the computing time of P-MDP is much slower with respect to AP-MDP as Figs. 8 shows. The x-axis denotes the computing time and the number of backups, and the y-axis denotes the cumulative number of cases evaluated. The red line and the blue line show computing time and the number of backups, respectively. Results of AP-MDP is indicated by the bold line and results of P-MDP (baseline) is indicated by the dotted line. In all settings, AP-MDP takes a shorter computing time and a smaller number of backups in most cases. The computing time of AP-MDP is shorter in 99, 99, 99, and 98 cases and the number of backup is smaller in 95, 89, 92, and 91 cases.



(a) Computing time ratio **(b)** The number of backups ratio

Fig. 8 Cumulative histograms of **a** computing time ratio (lower is better) and **b** the number of backups ratio (lower is better) of AP-MDP to P-MDP

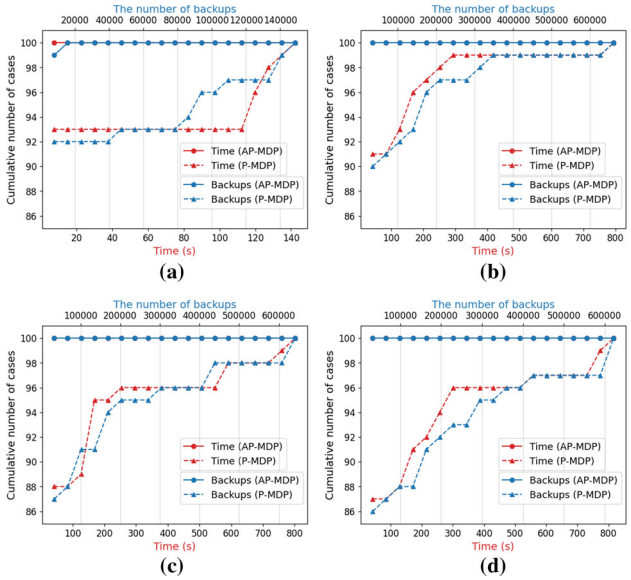


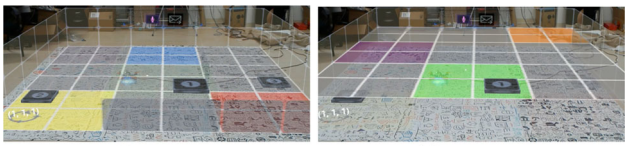
Fig. 9 Cumulative histograms of computing time and the number of backups of AP-MDP and P-MDP with **a** one object, **b** two objects, **c** three objects, and **d** four objects. The y-axis indicates the cumulative number of cases evaluated. We execute two algorithms with 100 random LTL tasks consisting of atomic propositions at the level 1

8 Robot experiments

In addition to the simulations described above, we also test our proposed method on two drones, one indoor and one outdoor. Video recordings of both experiments can be found at https://youtu.be/vwH2TQ2HEN8.

8.1 Indoor experiments with mixed reality

We use PiDrone (Brand et al., 2018) for our indoor experiments. The PiDrone is a quadcopter drone that is equipped with one downward-facing infrared sensor with a maximum range of 60cm to measure the drone's altitude, and one downward-facing camera for localization over a textured surface. The drone's flight space is a $3m \times 3m$ surface. We divide the space into a grid-based environment, as shown in Fig. 10, consisting of 2 floors, each with 9 rooms, and each room is a square made up of 4 cells (each cell is $50cm \times 50cm$). The action space for the drone in the grid-based environment is (*north, south, east, west, up, down*), where each action



(a) First floor **(b)** Second floor

Fig. 10 Figures of the two-floor environment for our drone demonstrations as viewed through the HoloLens, taken from our video

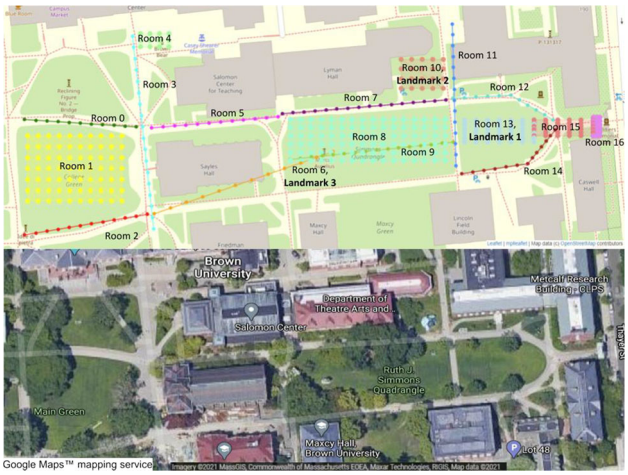


Fig. 11 Figures of the outdoor environment for our drone demonstration on Brown University's campus

changes the drone's location by 1 cell. We visualize the environment through mixed reality using a Microsoft HoloLens (Chen et al., 2015). Colored rooms and landmarks (boxes each with the size of 1 cell) to aid path planning and specify goal positions were set up in a Unity3D virtual environment running on the HoloLens.

In our experiments, the drone is given a natural language instruction through speech. This is converted using Google's speech-to-text, and then translated by our trained sequence-to-sequence model into an LTL formula to be solved by the AP-MDP framework in real time. The action sequence output by AP-MDP for the LTL expression is then used for the drone's navigation. The natural language commands were: *"Navigate to the red room", "Avoid landmark two until you have been to the blue room", "Move to the orange room then the purple room", "Go to landmark three then go to the yellow room."* Video recordings of the indoor experiments can be found at https://youtu.be/zjtMEGUmkd8.

8.2 Experiments in an outdoor environment

Motivated by the increased planning efficiency of AP-MDPs, we test our framework in an outdoor environment. We use Skydio R1 [2] for our outdoor experiment. The drone is equipped with a forward-facing color camera and 6 stereo camera pairs providing an omnidirectional view of the environment. We chose Skydio R1 for its robust obstacle avoidance capabilities and support for navigation in global coordinates. The drone's environment, as shown in Fig. 11, is composed of 447 nodes and 17 rooms, where each room is either a continuous region or line of nodes. Landmarks are defined as rooms, which are the higher level of abstraction. Under the Albers projection (Snyder, 1987), the area of the convex hull enclosing the environment is 6593.25 meters^2. The drone can move to adjacent nodes spaced 5 meters or less apart. At the lowest level, the action space is defined as `{GotoNode1,...,GotoNodeK}`, where `K` is the number of adjacencies for the most-connected node in the graph. At the room level, the action space is `{GotoRoom1,...,GotoRoomN}`, where `N` is the number of rooms in the environment.

In this experiment, language commands are given to a laptop with 4 processor cores and 8GB of RAM. The sequence-to-sequence model and AP-MDP framework are run onboard the laptop. Unlike the indoor experiment, AP-MDP outputs a sequence of global coordinates. These coordinates are wirelessly uploaded to Skydio R1 via HTTP. Due to safety purposes, challenges with speech-to-text in the outdoor environment, and to conserve the drone's battery life, the AP-MDP-planned coordinate sequence is cached before flight. To keep landmarks in the drone's field of view, the altitude is manually set for each path. Flying at lower altitudes introduces additional complex obstacles, such as trees, that are handled by Skydio R1's on-board motion planning. In effect, the AP-MDP-planned trajectory provides a higher-level plan that satisfies LTL constraints, while the drone's lower-level motion planner resolves local obstacles in the outdoor environment. The drone successfully followed plans for two commands: *"avoid landmark one until you have been to the second landmark"* and *"go to landmark three."* Visualizations of the drone's path are shown in Figs. 12 and 13, and planning results are shown in Table 1. These results demonstrate that the AP-MDP framework can resolve LTL specifications in the larger environment. A video recording of the outdoor experiment can be found at https://youtu.be/LhtQ7SiVcI8.

Additionally, we verified that avoid-class constraints can induce a modified path by testing the LTL constraints $\mathcal{F}$`(landmark_2)` and ¬`landmark_4` $\mathcal{U}$ `landmark_2`. In this experiment, input is an LTL formula and output is the AP-MDP-planned coordinate sequence. We performed these tests after the original outdoor experiment. Due to the COVID-19 pandemic, we did not test these trajectories on the drone. Visualizations of the drone's path are shown in Fig. 14 and planning results are shown in Table 1.

[2] Skydio R1: https://robots.ieee.org/robots/skydio1/

Fig. 12 Planned path for the command: *"avoid landmark one until you have been to the second landmark."* Circled points (left) correspond to drone-perspective images (right) in the outdoor environment

Fig. 13 Planned path for the command: *"go to landmark three."* Circled points (left) correspond to drone-perspective images (right) in the outdoor environment. The orange line denotes Landmark 3 (Room 6)

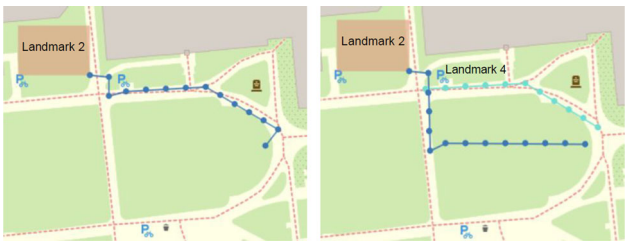


Fig. 14 Simulated drone trajectories for $\mathcal{F}$(`landmark_2`) (left) and ¬`landmark_4` $\mathcal{U}$ `landmark_2` (right), where the avoid predicate induces a modified path (right)

Table 1 Planning results for outdoor experiments

LTL	Planning Time (s)	Actions	Backups
¬`landmark_1` $\mathcal{U}$ `landmark_2`	164.90 ± 5.09	13	66080
$\mathcal{F}$(`landmark_3`)	230.14 ± 33.44	17	66048
$\mathcal{F}$(`landmark_2`)	158.6 ± 0.19	13	66080
¬`landmark_4` $\mathcal{U}$ `landmark_2`	158.07 ± 1.06	13	66080

9 Conclusion

This paper introduces a novel approach to combine the handling of non-Markovian task specifications in large environments by grounding complex language to LTL expressions and then decomposing tasks within an abstraction hierarchy to plan efficiently at higher levels where possible. We show that planning with abstractions allows the robot to correctly reach the goal location more efficiently, in terms of computing time and backups required, in over 95% of tasks in a small planning problem and over 99% of tasks in a larger planning problem. In a cleanup world, AP-MDP is more efficient in over 91 of tasks. We also show that this method of abstraction can handle LTL task specifications. Moreover, we present the largest existing dataset of natural language commands mapped to LTL expressions at different levels of abstraction. We demonstrate our approach in both indoor and outdoor environments with two drones that navigate to the goal location along a correct path when given a human-uttered command.

While the language grounding model works fairly well to translate language to LTL formulae, it cannot fully handle expressions unseen during training and cannot always deal with simple changes in word-ordering and variations in the language. Future work in this direction can explore compositional models that can handle a wide range of expressions by learning to compose subparts together and then execute the required actions. Future work in the hierarchical setup can explore models that go beyond fixed hierarchies and state abstractions. If the hierarchies can be learned with model-learning methods on the fly, this will enable generalization to unseen environments and other domains.

Acknowledgements This work is supported in part by the National Science Foundation under grant numbers IIS-1637614 and IIS-1652561, and the National Aeronautics and Space Administration under grant number NNX16AR61G. This work is supported in part by Samsung Research Funding & Incubation Center of Samsung Electronics under Project Number SRFC-IT2002-05.

References

Arumugam, D., Karamcheti, S., Gopalan, N., Wong, L. L., Tellex, S. (2017). Accurately and efficiently interpreting human-robot instructions of varying granularities. arXiv preprint arXiv:1704.06616.

Bhatia, A., Kavraki, L. E., Vardi, M. Y. (2010). Sampling-based motion planning with temporal goals. 2010 IEEE International Conference on Robotics and Automation.

Boteanu, A., Howard, T., Arkin, J., Kress-Gazit, H. (2016). A model for verifiable grounding and execution of complex natural language instructions. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

Brand, I., Roy, J., Ray, A., Oberlin, J., Oberlix, S. (2018). Pidrone: An autonomous educational drone using raspberry pi and python. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

Büchi, J. R. (1990). On a decision method in restricted second order arithmetic. *The Collected Works of J* (pp. 425–435). Richard Büchi: Springer.

Chen, H., Lee, A. S., Swift, M., Tang, J. C. (2015). 3d collaboration method over hololens™ and skype™ end points. In Proc. of the 3rd International Workshop on Immersive Media Experiences.

Cho, K., Suh, J., Tomlin, C. J., & Oh, S. (2017). Cost-aware path planning under co-safe temporal logic specifications. *IEEE Robotics and Automation Letters, 2*(4), 2308–2315.

Dasgupta, I., Guo, D., Stuhlmüller, A., Gershman, S. J., Goodman, N. D. (2018). Evaluating compositionality in sentence embeddings. CoRR abs/1802.04302.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research, 13*, 227–303.

Ding, X., Smith, S. L., Belta, C., & Rus, D. (2014). Optimal control of markov decision processes with linear temporal logic constraints. *IEEE Transactions on Automatic Control, 59*(5), 1244–1257.

Ding, X. C., Smith, S. L., Belta, C., Rus, D. (2011). MDP optimal control under temporal logic constraints. In IEEE Conference on Decision and Control and European Control Conference (CDC-ECC).

Duret-Lutz, A. (2022). Spot's temporal logic formulas. Tech. rep., https://spot.lrde.epita.fr/tl.pdf.

Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L. (2016). Spot 2.0 — a framework for LTL and ω-automata manipulation. In Proc. of the International Symposium on Automated Technology for Verification and Analysis (ATVA'16), Springer, Lecture Notes in Computer Science.

Fainekos, G. E., Girard, A., Kress-Gazit, H., & Pappas, G. J. (2009). Temporal logic motion planning for dynamic robots. *Automatica, 45*(2), 343–352. https://doi.org/10.1016/j.automatica.2008.08.008

Finucane, C., Jing, G., Kress-Gazit, H. (2010). Ltlmop: Experimenting with language, temporal logic and robot control. In 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems.

Fu, J., Topcu, U. (2014). Probably approximately correct MDP learning and control with temporal logic constraints. arXiv preprint arXiv:1404.7073.

Gopalan, N., desJardins, M., Littman, M. L., MacGlashan, J., Squire, S., Tellex, S., Winder, J., Wong, L. L. (2017). Planning with abstract markov decision processes. In ICAPS.

Gopalan, N., Arumugam, D., Wong, L., & Tellex, S. (2018). *Sequence-to-sequence language grounding of non-markovian task specifications*. In Robotics: Science and Systems.

Kasenberg, D., Scheutz, M. (2017) Interpretable apprenticeship learning with temporal logic specifications. In IEEE Conference on Decision and Control.

Kingma, D. P., Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kloetzer, M., Mahulea, C. (2016). Multi-robot path planning for syntactically co-safe ltl specifications. In 2016 13th International Workshop on Discrete Event Systems (WODES), pp. 452–458, https://doi.org/10.1109/WODES.2016.7497887.

Koehn, P., Knowles, R. (2017). Six challenges for neural machine translation. arXiv preprint arXiv:1706.03872.

Konidaris, G. (2016). Constructing abstraction hierarchies using a skill-symbol loop. In Proc. of the International Joint Conference on Artificial Intelligence.

Konidaris, G., Kaelbling, L. P., & Lozano-Perez, T. (2018). From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research, 61*, 215–289.

Kress-Gazit, H., Fainekos, G. E., & Pappas, G. J. (2008). Translating structured english to robot controllers. *Advanced Robotics, 22*(12), 1343–1359.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., & Tenenbaum, J. (2016). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 29*. USA: Curran Associates Inc.

Lake, B. M., Baroni, M. (2017). Still not systematic after all these years: On the compositional skills of sequence-to-sequence recurrent networks. arXiv preprint arXiv:1711.00350.

Lignos, C., Raman, V., Finucane, C., Marcus, M., & Kress-Gazit, H. (2015). Provably correct reactive control from natural language. *Autonomous Robots, 38*(1), 89–105.

Liu, X., Fu, J. (2018). Compositional planning in markov decision processes: Temporal abstraction meets generalized logic composition. arXiv preprint arXiv:1810.02497.

MacGlashan, J., Babes-Vroman, M., & desJardins, M., Littman, M., Muresan, S., Squire, S., Tellex, S., Arumugam, D., Yang, L. (2015). *Grounding english commands to reward functions*. In Robotics: Science and Systems.

MacGlashan, J., Babes-Vroman, M., & desJardins, M., Littman, M. L., Muresan, S., Squire, S., Tellex, S., Arumugam, D., Yang, L. (2015). *Grounding english commands to reward functions*. In Robotics: Science and Systems.

Manna, Z., Pnueli, A. (1990) A hierarchy of temporal properties. In Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing, Association for Computing Machinery, pp. 377–410.

McMahon, J., Plaku, E. (2014). Sampling-based tree search with discrete abstractions for motion planning with dynamics and temporal logic. In Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems.

Oh, Y., Cho, K., Choi, Y., & Oh, S. (2017). Robust multi-layered sampling-based path planning for temporal logic-based missions. *In IEEE Conference on Decision and Control*. https://doi.org/10.1109/CDC.2017.8263891

Oh, Y., Patel, R., Nguyen, T., Huang, B., Pavlick, E., Tellex, S. (2019). Planning with state abstractions for non-markovian task specifications. In Proceedings of Robotics: Science and Systems, FreiburgimBreisgau, Germany, https://doi.org/10.15607/RSS.2019.XV.059.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A. (2017). Automatic differentiation in pytorch.

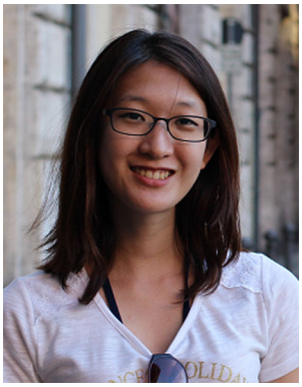
Sadigh, D., Kim, E. S., Coogan, S., Sastry, S. S., & Seshia, S. A. (2014). A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. *In IEEE Conference on Decision and Control*. https://doi.org/10.1109/CDC.2014.7039527

Snyder, J. P. (1987). Map projections–A working manual, vol 1395, US Government Printing Office, pp. 98–103.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research, 15*(1), 1929–1958.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence, 112*(1–2), 181–211.

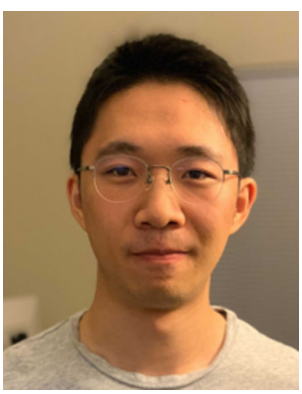
Wolff, E. M., Topcu, U., Murray, R. M. (2012). Robust control of uncertain markov decision processes with temporal logic specifications. In IEEE Conference on Decision and Control.



Yoonseon Oh is currently an assistant professor, in the department of Electronic Engineering, Hanyang University, Republic of Korea. She received the Ph.D. degree in the Department of Electrical and Computer Engineering in 2018 and the B.S. degree in the Department of Electrical and Electronics Engineering in 2011 from Seoul National University, Seoul, Korea. From 2019 to 2021, she was a Senior Researcher and Researcher at Center for Robotics Research, in Korea Institute of Science and Technology. From 2018 to 2019, she was a Postdoctoral Researcher in the Department of Computer Science at Brown University, RI, USA. Her research interests include robotics, task learning and planning, and human-robot interaction.

Roma Patel is a Ph.D. student at Brown University advised by Ellie Pavlick. Her research interest is building models for natural language understanding and knowledge representations.

Thao Nguyen is a Ph.D. student at Brown University's Humans to Robots Lab, advised by Prof. Stefanie Tellex. Her research interests include language grounding, planning, and robot learning.



Baichuan Huang is a Ph.D. student in Computer Science at Rutgers University. He is currently focusing on robot manipulation and planning. From 2017 to 2019, he was a master student in Computer Science at Brown University, worked on drone and Mixed Reality.

Matthew Berg is graduating from Brown University with an Sc.B. in Computer Science. He worked in the Humans To Robots Laboratory and is joining Amazon Robotics. As an undergraduate researcher, he contributed to human-robot language interfaces, mixed reality control interfaces, a long-range planning system, and a computational model for teaching robots how to follow social norms.

Ellie Pavlick recieved her Ph.D. in Computer and Information Science from University of Pennsylvania in 2017. In 2012, she received a Bachelor of Arts in Economics from Johns Hopkins University and a Bachelor of Music is Saxophone Performance from the Peabody Conservatory. Ellie's current research is in Natural Language Processing, specifically on computational models of semantics and pragmatics which emulate human inferences.

Stefanie Tellex is the Joukowsky Family Associate Professor of Computer Science and Assistant Professor of Engineering at Brown University. Her group, the Humans To Robots Lab, creates robots that seamlessly collaborate with people to meet their needs using language, gesture, and probabilistic inference, aiming to empower every person with a collaborative robot.