Planning with State Abstractions for Non-Markovian Task Specifications

Yoonseon Oh, Roma Patel, Thao Nguyen, Baichuan Huang, Ellie Pavlick, and Stefanie Tellex

***Abstract*—Often times, we specify tasks for a robot using temporal language that can also span different levels of abstraction. The example 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 Linear Temporal Logic (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 tasks, and this number only increases as the size of the environment domain increases. 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 a drone, demonstrating that our approach enables a robot to efficiently solve temporal commands at different levels of abstraction.**

I. 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 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.

The authors are with the Brown University Department of Computer Science, 115 Waterman Street, Providence, RI 02912. Email: {yoonseon_oh, romapatel, thao_nguyen3, baichuan_huang, ellie_pavlick}@brown.edu, stefie10@cs.brown.edu

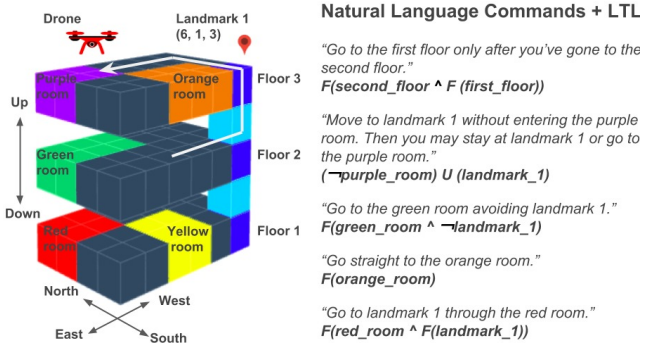


Fig. 1. Our environment is a gridworld with three floors, each consisting of rooms that consist of grid cells. The white arrow shows an example path the drone can take in the environment. We also include sample natural language commands (and their LTL formulae) that the drone successfully executed.

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) [1]. However, these models very quickly become intractable as the state space of the world grows larger [2, 3]. Planning with *abstractions* in a hierarchical structure [2, 3, 4, 5], either by using an Abstract Markov Decision Process (AMDP) [2] or with options [3, 4, 5] 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 [6]. Separately, as shown in Fig. 1, non-Markovian natural language commands can be mapped to linear temporal logic (LTL) formulae [7, 8, 9, 10] to allow efficient planning with an MDP, given the corresponding LTL task specifications [11, 12, 13, 14, 15, 16, 17]. 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 this paper, we introduce the *Abstract Product MDP (AP-MDP)* framework to combine the benefits of LTL and

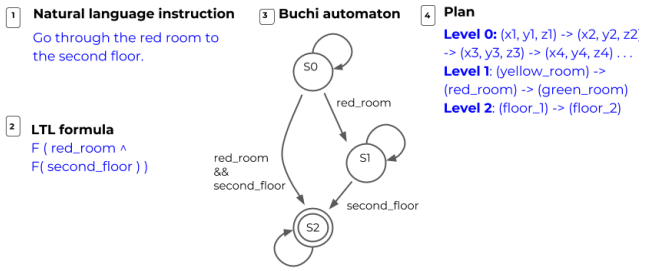


Fig. 2. Complete pipeline for the translation of a natural language instruction to an LTL formula, then to a Büchi automaton, and to a plan that gives us action sequences to correctly reach the goal location specified by the task.

AMDP, thus enabling a robot to interpret non-Markovian commands at different levels of abstraction. There is previous work in planning for LTL tasks using options [18]. 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 [2]. 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 [19]. 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.

We evaluate our approach by reporting the performance of AP-MDP in simulation and on a drone platform. 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 a drone.

II. 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 [11, 12, 13, 15, 16]. Some previous works model dynamic systems of agents as MDPs and developed methods to generate a control policy that satisfies LTL constraints [11, 12]. 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 [14] 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 [2, 3, 4, 20] are commonly used. The options framework [3, 4], for example, models temporally abstract macro-actions as options that can be adopted to build abstraction hierarchies. Similarly, AMDPs [2] 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 [21, 22, 23, 24] 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 [22, 23, 24]. Other works have focused on grounding natural language to LTL expressions [9, 10, 17] 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 [18]. However, the AMDP framework [2] 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.

III. 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 gets a high reward when the task is accomplished. 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 [7, 19], and then an AMDP [2] to decompose the problem into a series of smaller, and hence easier to solve, subproblems. Section IV defines LTL and the variants of MDPs that our model relies on, while section V 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.

IV. 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 and its syntax, explain the transformation of an LTL expression to a Büchi automaton and further to an MDP.

A. 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: $\phi ::= \pi \mid \neg\phi \mid \phi \wedge \varphi \mid \phi \vee \varphi \mid \mathcal{G}\phi \mid \mathcal{F}\phi \mid \phi\ \mathcal{U}\varphi$, 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".

B. Linear temporal logic to 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 Büchi automaton [19] — a deterministic automaton, that differs from the general notion of automata in that it accepts infinite traces represented by the input LTL formula. 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: (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 iff its run r satisfies the relationship $\lim(r) \cap \mathbf{F} \neq \emptyset$, that is, the language $L(\mathcal{B})$ is non-empty if at least one final state is reached.

C. 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 [13] 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.

D. 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 [15, 16], 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) $L_p((s,q)) = q$,

E. Abstract Markov Decision Processes

An Abstract Markov Decision Process [2] (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.

V. 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.

A. 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.

B. 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 section V-C to highlight the process of how decomposed subproblems are solved for the task specification in question. Section V-D then explains how any problem can be decomposed into component subproblems and section V-E presents the pseudocode for the algorithm for this process. The language grounding component of the system is discussed in V-F and finally, V-G describes the functioning of the end-to-end system.

C. Example Problem

Consider the example in Fig. 3. This figure shows the DBA for the LTL task specification $\phi = \mathcal{F}(\alpha^0_0 \wedge \mathcal{F}(\alpha^1_0 \wedge \mathcal{F}\alpha^1_1))$ and we can see that the atomic proposition α^0_0 is in level 0 of the abstraction hierarchy, while α^1_0 and α^1_1 are 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 the logical expression on the edge is satisfied by the state. The reward

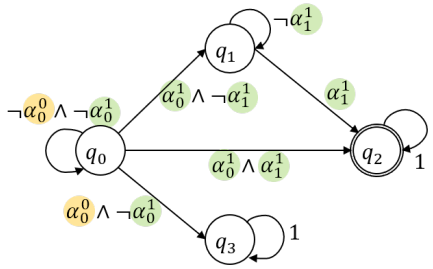


Fig. 3. Deterministic Büchi automaton. Atomic propositions in yellow circles correspond to those in level 0 and atomic propositions in green circles correspond to those in level 1. The transitions of the automaton refer to constraints over the propositions that are satisfied on taking that path.

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 $\hat{\mathcal{M}}_1$. This has atomic propositions related to level 1, therefore $\hat{\mathcal{M}}_1$ can be formulated at a higher level of abstraction, that is, level 1 ($\hat{\mathcal{M}}_1 = (\hat{S}^1,\ \hat{A}^1,\ \hat{T}^1,\ \hat{s}_0^1,\ AP,\ \mathcal{L}^1,\ R^1)$), 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.

D. Subproblem Decomposition

In general, there are n_ρ paths in a Büchi automaton from the initial state to the accepting state. AP-MDPs decompose the problem into n_ρ subproblems each denoted by $\mathcal{P}_{\rho_i}$, which accomplish the LTL task while following the path ρ_i.

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

$$R_j = \begin{cases} \gamma_{goal}, & \text{if } f(\hat{q}^i_j, \hat{q}^i_{j+1}, s, s') = true, \\ \gamma_{stay}, & \text{else if } f(\hat{q}^i_j, \hat{q}^i_j, s, s') = false, \\ \gamma, & \text{otherwise,} \end{cases} \tag{1}$$

where $\gamma_{goal} \gg 1$, $\gamma_{stay} \ll 0$, and γ is a small negative value.

In this way, AP-MDPs can consist of $(\sum_{i=1}^{n_\rho} n_i)$ AL-MDPs. When we denote the plan for $\mathcal{P}_{\rho_i}$ as $(s_{seq}, a_{seq})^{\rho_i}$, the plan for the LTL task is the shortest sequence $(s_{seq}, a_{seq})^*$, where s_{seq} is the state sequence and a_{seq} the is action sequence.

E. 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

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** $\rho_i \in Paths$ **do**
8: Initialize s_0
9: Initialize the plan $(s_{seq},\ a_{seq})^{\rho_i}$
10: **for** j in $\{0, \cdots, n_i - 1\}$ **do**
11: *goal condition* $\leftarrow f(\hat{q}^i_j, \hat{q}^i_{j+1}, s, s') = true$
12: *stay condition*$\leftarrow f(\hat{q}^i_j, \hat{q}^i_j, s, s') = true$
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: $\pi \leftarrow \hat{\mathcal{M}}_j.Solve()$
16: $ss, aa \leftarrow \hat{\mathcal{M}}_j.Plan(\pi,\ s_0)$
17: $(s_{seq},\ a_{seq})^{\rho_i} \leftarrow (s_{seq},\ a_{seq})^{\rho_i} \cup (ss, aa)$
18: $s_0 \leftarrow s_{seq}(end)$
19: **end for**
20: **if** $length(s_{seq}) < l^*$ **then**
21: $(s_{seq}, a_{seq})^* \leftarrow (s_{seq}, a_{seq})^{\rho_i}$
22: **end if**
23: **end for**

formula into a DBA using an existing package called Spot2 (line 4) [25]. 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. 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 policy and plan of AL-MDPs with a solver of the AMDP (lines 15-16). We then select the best plan which has the minimum number of actions (lines 20-22).

F. 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.

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

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

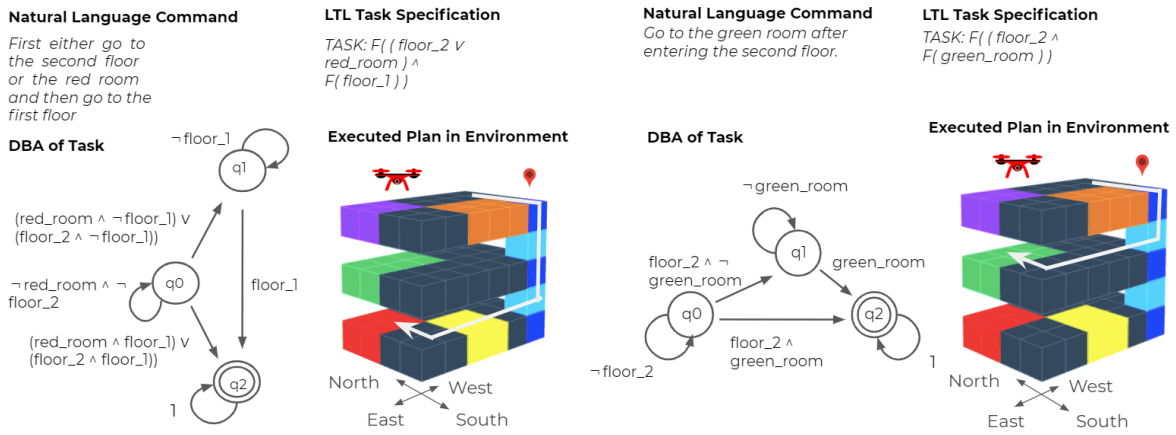


Fig. 4. Examples, left and right, tested in simulation. In each example, a natural language instruction is converted to an LTL expression, then to a corresponding AP-MDP to find a policy. An agent then executes the policy in the specified environment to reach the correct goal state through the desired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

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.

2) Sequence-to-sequence model: As in Gopalan et al. [17], 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 [26] and trained for 10 epochs over our corpus, with a learning rate of 0.001 using the Adam optimizer [27]. We used a dropout of 0.8 as a regularizer [28].

G. Planning for an LTL task

Once a natural language command is translated into an LTL formula, it is then converted into a Büchi automaton with multiple paths from the initial state to the accepting state. Each path is represented and solved with an AL-MDP.

VI. Experiments

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

A. Environment Setup

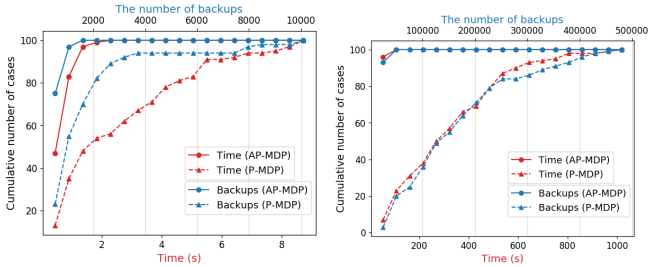
For simulations, we consider two 3D grid worlds ($\mathcal{E}_1$ and $\mathcal{E}_2$) of size $6\times4\times3$ and $30\times20\times6$, respectively. The smaller world $\mathcal{E}_1$ has three floors, each comprised of six rooms, each the size of 2×2 grid cells. The larger world $\mathcal{E}_2$ has 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.

B. 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:

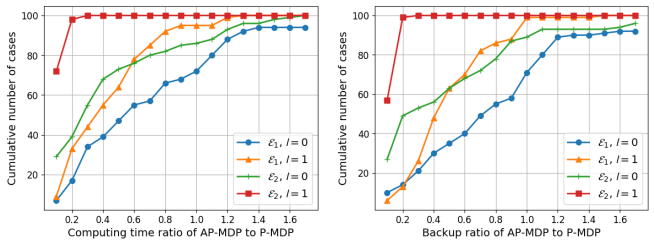
1) ϕ_1=$\mathcal{F}$ ((`floor_2` $\vee$ `red_room`) $\wedge$ $\mathcal{F}$(`floor_1`))
   ("First either go to the second floor or the red room, and then go to the first floor")
2) $\phi_2 = \mathcal{F}$(`floor_2` $\wedge$ $\mathcal{F}$(`green_room`))
   ("Go to the green room after entering the second 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 $\rho_0 = q_0q_2$ 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 $\rho_1 = q_0q_1q_2$, 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



(a) (b)

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.



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

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.

we can satisfy ϕ_1 with only two actions with ρ_0, the final solution is a plan for ρ_0.

For task ϕ_2, there exists an infeasible path among paths in the automaton. The first AL-MDP in $\rho_0 = q_0 q_2$ has goal and stay conditions of $(\texttt{floor\_2} \wedge \texttt{green\_room})$ and $(\neg\texttt{floor\_2})$, 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 $\rho_1 = q_0 q_1 q_2$. The first AL-MDP has a goal condition of $(\texttt{floor\_2} \wedge \neg\texttt{green\_room})$ and a stay condition of $(\neg\texttt{floor\_2})$. The second AL-MDP has a goal condition of $(\texttt{green\_room})$ with a stay condition of $(\neg\texttt{green\_room})$. The planned path is shown in Fig. 4.

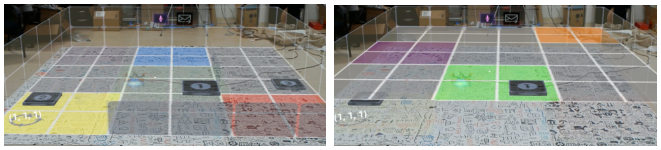
C. *Efficiency*

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 section IV-D, 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 $\{\mathcal{F}a,\ \mathcal{F}(a \wedge \mathcal{F}b),\ \mathcal{F}(a \wedge \mathcal{F}(b \wedge \mathcal{F}c)),\ \mathcal{F}a \wedge \mathcal{F}b,\ \neg a\ \mathcal{U}\ b\}$, 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 Fig. 5 and Fig. 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.

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



(a) First floor (b) Second floor

Fig. 7. Figures of the two-floor environment for our drone demonstrations as viewed through the HoloLens, taken from our video.

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.

D. Language grounding results

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 [17, 29, 30] for neural sequence-to-sequence models that cannot handle compositionality and the ability of models to learn meaning representations for given natural language sentences [31]. 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.

E. Robot Experiments

In addition to the simulations described above, we also test our proposed method on a drone. The PiDrone [32] 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. 7, 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 changes the drone's location by 1 cell. We visualize the environment through mixed reality using a Microsoft HoloLens [33]. 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 demonstration, 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 drone demonstrations can be found at https://youtu.be/zjtMEGUmkd8.

VII. 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 environment and over 99% of tasks in a larger environment. 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 with a PiDrone that navigates 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 AMDP transition hierarchies can be learned with model-learning methods on the fly, this will enable generalization to unseen environments and the ability to handle and properly execute a plan for a wider range of commands.

VIII. Acknowledgments

The authors would like to thank Nakul Gopalan for his insightful comments and edits. This work is supported by the National Science Foundation under grant numbers IIS-1637614 and IIS-1652561, and the National Aeronautics and Space Administration under grant number NNX16AR61G.

References

[1] J. MacGlashan, M. Babes-Vroman, M. desJardins, M. L. Littman, S. Muresan, S. Squire, S. Tellex, D. Arumugam,

and L. Yang, "Grounding english commands to reward functions." in *Robotics: Science and Systems*, 2015.

[2] N. Gopalan, M. desJardins, M. L. Littman, J. MacGlashan, S. Squire, S. Tellex, J. Winder, and L. L. Wong, "Planning with abstract markov decision processes," in *ICAPS*, 2017.

[3] G. Konidaris, "Constructing abstraction hierarchies using a skill-symbol loop," in *Proc. of the International Joint Conference on Artificial Intelligence*, 2016.

[4] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *Journal of Artificial Intelligence Research*, vol. 61, pp. 215–289, 2018.

[5] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[6] D. Arumugam, S. Karamcheti, N. Gopalan, L. L. Wong, and S. Tellex, "Accurately and efficiently interpreting human-robot instructions of varying granularities," *arXiv preprint arXiv:1704.06616*, 2017.

[7] C. Finucane, G. Jing, and H. Kress-Gazit, "Ltlmop: Experimenting with language, temporal logic and robot control," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[8] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Translating structured english to robot controllers," *Advanced Robotics*, vol. 22, no. 12, pp. 1343–1359, 2008.

[9] C. Lignos, V. Raman, C. Finucane, M. Marcus, and H. Kress-Gazit, "Provably correct reactive control from natural language," *Autonomous Robots*, vol. 38, no. 1, pp. 89–105, 2015.

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

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

[12] X. Ding, S. L. Smith, C. Belta, and D. Rus, "Optimal control of markov decision processes with linear temporal logic constraints," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1244–1257, 2014.

[13] J. Fu and U. Topcu, "Probably approximately correct MDP learning and control with temporal logic constraints," *arXiv preprint arXiv:1404.7073*, 2014.

[14] D. Kasenberg and M. Scheutz, "Interpretable apprenticeship learning with temporal logic specifications," in *IEEE Conference on Decision and Control*, 2017.

[15] E. M. Wolff, U. Topcu, and R. M. Murray, "Robust control of uncertain markov decision processes with temporal logic specifications," in *IEEE Conference on Decision and Control*, 2012.

[16] D. Sadigh, E. S. Kim, S. Coogan, S. S. Sastry, and S. A. Seshia, "A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications," in *IEEE Conference on Decision and Control*, 2014.

[17] N. Gopalan, D. Arumugam, L. Wong, and S. Tellex, "Sequence-to-sequence language grounding of non-markovian task specifications," in *Robotics: Science and Systems*, 2018.

[18] X. Liu and J. Fu, "Compositional planning in markov decision processes: Temporal abstraction meets generalized logic composition," *arXiv preprint arXiv:1810.02497*, 2018.

[19] J. R. Büchi, "On a decision method in restricted second order arithmetic," in *The Collected Works of J. Richard Büchi*. Springer, 1990, pp. 425–435.

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

[21] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343 – 352, 2009.

[22] J. McMahon and E. Plaku, "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*, 2014.

[23] K. Cho, J. Suh, C. J. Tomlin, and S. Oh, "Cost-aware path planning under co-safe temporal logic specifications," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2308–2315, 2017.

[24] Y. Oh, K. Cho, Y. Choi, and S. Oh, "Robust multi-layered sampling-based path planning for temporal logic-based missions," in *IEEE Conference on Decision and Control*, 2017.

[25] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu, "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)*, ser. Lecture Notes in Computer Science. Springer, 2016.

[26] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[29] B. M. Lake and M. Baroni, "Still not systematic after all these years: On the compositional skills of sequence-to-sequence recurrent networks," *arXiv*

preprint arXiv:1711.00350, 2017.

[30] P. Koehn and R. Knowles, "Six challenges for neural machine translation," *arXiv preprint arXiv:1706.03872*, 2017.

[31] I. Dasgupta, D. Guo, A. Stuhlmüller, S. J. Gershman, and N. D. Goodman, "Evaluating compositionality in sentence embeddings," *CoRR*, vol. abs/1802.04302, 2018. [Online]. Available: http://arxiv.org/abs/1802.04302

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

[33] H. Chen, A. S. Lee, M. Swift, and J. C. Tang, "3d collaboration method over hololens and skype end points," in *Proc. of the 3rd International Workshop on Immersive Media Experiences*, 2015.