LTL2Action: Generalizing LTL Instructions for Multi-Task RL

Pashootan Vaezipoor [* 1 2] **Andrew C. Li** [* 1 2] **Rodrigo Toro Icarte** [1 2] **Sheila McIlraith** [1 2 3]

Abstract

We address the problem of teaching a deep reinforcement learning (RL) agent to follow instructions in multi-task environments. Instructions are expressed in a well-known formal language – *linear temporal logic* (LTL) – and can specify a diversity of complex, temporally extended behaviours, including conditionals and alternative realizations. Our proposed learning approach exploits the compositional syntax and the semantics of LTL, enabling our RL agent to learn task-conditioned policies that generalize to new instructions, not observed during training. To reduce the overhead of learning LTL semantics, we introduce an environment-agnostic LTL pretraining scheme which improves sample-efficiency in downstream environments. Experiments on discrete and continuous domains target combinatorial task sets of up to $\sim 10^{39}$ unique tasks and demonstrate the strength of our approach in learning to solve (unseen) tasks, given LTL instructions.

1. Introduction

A long-standing aspiration of artificial intelligence is to build agents that can understand and follow human instructions to solve problems (McCarthy et al., 1960). Recent advances in deep and reinforcement learning (RL) have made it possible to learn a policy that decides the next action conditioned on the current observation and a natural language instruction. Given enough training data, the learned policy will show some degree of generalization to unseen instructions (e.g., Hermann et al., 2017; Oh et al., 2017; Chaplot et al., 2018; Yu et al., 2018; Co-Reyes et al., 2019; Jiang et al., 2019; Luketina et al., 2019). Unfortunately, such approaches do not scale well because they require (for every possible environment) manually building a large training set comprised of natural language instructions with their corresponding reward functions.

Motivated by this observation, recent works have explored using structured or formal languages (instead of natural language) to instruct RL agents. Such languages offer several desirable properties for RL, including unambiguous semantics, and compact compositional syntax that enables RL practitioners to (automatically) generate massive training data to teach RL agents to follow instructions. Examples of such languages include policy sketches (Andreas et al., 2017), task graphs (Sohn et al., 2018), procedural programs (Sun et al., 2019), declarative programs (Denil et al., 2017), reward machines (Toro Icarte et al., 2020), and temporal logic (Leon et al., 2020). Many of these methods exploit compositional syntax to decompose instructions into smaller subtasks that are solved independently, without consideration for the subtasks that follow. This can lead to subtask policies that are individually optimal, but when combined are suboptimal with respect to the instructions as a whole. We refer to these as *myopic approaches*.

In this work, we use *linear temporal logic (LTL)* (Pnueli, 1977) over a domain-specific vocabulary (e.g., *have-coffee*) to instruct RL agents to learn policies that generalize well to unseen instructions without compromising optimality guarantees – in contrast to typical myopic methods. LTL is an expressive formal language that combines temporal modalities such as *eventually*, *until*, and *always* with binary predicates that establish the truth or falsity of an event or property (e.g., *have-coffee*), composed via logical connectives to support specification of goal sequences, partial-order tasks, safety constraints, and much more. Our learning algorithm exploits a semantics-preserving rewriting operation, called *LTL progression*, that allows the agent to identify aspects of the original instructions that remain to be addressed in the context of an evolving experience. This enables learning policies in a non-myopic manner, all the while preserving optimality guarantees and supporting generalization.

Our approach is realized in a deep RL setting, exploiting event detectors to recognize domain vocabulary. We encode LTL instructions using an LSTM, GRU, or a Graph Neural Network (GNN). To reduce the overhead of learning

[*]Equal contribution [1]Department of Computer Science, University of Toronto [2]Vector Institute for Artificial Intelligence [3]Schwartz Reisman Institute for Technology and Society. Correspondence to: Pashootan Vaezipoor <pashootan@cs.toronto.edu>, Andrew Li <andrewli@cs.toronto.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

LTL semantics, we introduce an environment-agnostic LTL pretraining scheme. We evaluate our approach on discrete and continuous domains. Our contributions are as follows:

- We propose a novel approach for teaching RL agents to follow LTL instructions that has theoretical advantages over existing RL methods employing LTL instructions (Kuo et al., 2020; Leon et al., 2020). This leads to better generalization performance in our experiments.
- We show that encoding LTL instructions via a neural architecture equipped with LTL progression yielded higher reward policies relative to a myopic approach. Out of the neural architectures GNN offered better generalization compared to LSTM and GRU.
- Lastly, we demonstrate that applying an environment-agnostic LTL pretraining scheme improves sample efficiency on downstream tasks.

2. Reinforcement Learning

RL agents learn optimal behaviours by interacting with an environment. Usually, the environment is modelled as a *Markov Decision Process (MDP)*. An MDP is a tuple $\mathcal{M} = \langle S, T, A, \mathbb{P}, R, \gamma, \mu \rangle$, where S is a finite set of *states*, $T \subseteq S$ is a finite set of *terminal states*, A is a finite set of *actions*, $\mathbb{P}(s'|s,a)$ is the *transition probability distribution*, $R : S \times A \times S \to \mathbb{R}$ is the *reward function*, γ is the *discount factor*, and μ is the *initial state distribution*.

The interactions with the environment are divided into *episodes*. At the beginning of an episode, the environment is set at some initial state $s_0 \in S$ sampled from μ. Then, at time step t, the agent observes the current state $s_t \in S$ and executes an action $a_t \in A$ according to some *policy* $\pi(a_t|s_t)$ – which is a probability distribution from states to actions. In response, the environment returns the next state s_{t+1} sampled from $\mathbb{P}(s_{t+1}|s_t, a_t)$ and an immediate reward $r_t = R(s_t, a_t, s_{t+1})$. This process then repeats until reaching a terminal state (starting a new episode). The agent's objective is to learn an *optimal policy* $\pi^*(a|s)$ that maximizes the expected discounted return $\mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| S_t = s\right]$ when starting from any state $s \in S$ and time step t.

3. Multitask Learning with LTL

In order to instruct RL agents using language, the first step is to agree upon a common vocabulary between us and the agent. In this work, we use a finite set of propositional symbols $\mathcal{P}$ as the vocabulary, representing high-level events or properties (henceforth "events") whose occurrences in the environment can be detected by the agent. For instance, in a smart home environment, $\mathcal{P}$ could include events such as opening the living room window, activating the fan, turning on/off the stove, or entering the living room. Then, we use LTL to compose temporally-extended occurrences of these events into instructions. For example, two possible instructions that can be expressed in LTL (but described here in plain English) are (1) "Open the living room window and activate the fan in any order, then turn on the stove" and (2) "Open the living room window but don't enter the living room until the stove is turned off".

In this section, we discuss how to specify instructions using LTL and automatically transform those instructions into reward functions, we formally define the RL problem of learning a policy that generalizes to unseen LTL instructions.

3.1. Linear Temporal Logic (LTL)

LTL extends propositional logic with two temporal operators: $\bigcirc$ (*next*) and U (*until*). Given a finite set of propositional symbols $\mathcal{P}$, the syntax of an LTL formula is defined as follows (Baier & Katoen, 2008):

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \psi \mid \bigcirc \varphi \mid \varphi \,\mathsf{U}\, \psi \quad \text{where } p \in \mathcal{P}$$

In contrast to propositional logic, LTL formulas are evaluated over sequences of observations (i.e., *truth assignments* to the propositional symbols in $\mathcal{P}$). Intuitively, the formula $\bigcirc\varphi$ (*next* φ) holds if φ holds at the next time step and $\varphi \,\mathsf{U}\, \psi$ (φ *until* ψ) holds if φ holds until ψ holds.

Formally, the truth value of an LTL formula is determined relative to an infinite sequence of truth assignments $\sigma = \langle \sigma_0, \sigma_1, \sigma_2, \ldots \rangle$ for $\mathcal{P}$, where $p \in \sigma_i$ iff proposition $p \in \mathcal{P}$ holds at time step i. Then, σ *satisfies* φ at time $i \geq 0$, denoted by $\langle \sigma, i \rangle \models \varphi$, as follows:

- $\langle \sigma, i \rangle \models p$ iff $p \in \sigma_i$, where $p \in \mathcal{P}$
- $\langle \sigma, i \rangle \models \neg\varphi$ iff $\langle \sigma, i \rangle \not\models \varphi$
- $\langle \sigma, i \rangle \models (\varphi \wedge \psi)$ iff $\langle \sigma, i \rangle \models \varphi$ and $\langle \sigma, i \rangle \models \psi$
- $\langle \sigma, i \rangle \models \bigcirc\varphi$ iff $\langle \sigma, i+1 \rangle \models \varphi$
- $\langle \sigma, i \rangle \models \varphi \,\mathsf{U}\, \psi$ iff there exists j such that $i \leq j$ and $\langle \sigma, j \rangle \models \psi$, and $\langle \sigma, k \rangle \models \varphi$ for all $k \in [i, j)$

A sequence σ is then said to *satisfy* φ iff $\langle \sigma, 0 \rangle \models \varphi$.

Any LTL formula can be define in terms of $p \in \mathcal{P}$, $\wedge$ (*and*), $\neg$ (*negation*), $\bigcirc$ (*next*), and U (*until*). However, from these operators, we can also define the Boolean operators $\vee$ (*or*) and $\to$ (*implication*), and the temporal operators $\Box$ (*always*) and $\Diamond$ (*eventually*), where $\langle \sigma, 0 \rangle \models \Box\varphi$ if φ always holds in σ, and $\langle \sigma, 0 \rangle \models \Diamond\varphi$ if φ holds at some point in φ.

As an illustrative example, consider the MiniGrid (Chevalier-Boisvert et al., 2018) environment in Figure 1. There are two rooms – one with a blue and a red square, and one with a blue and a green square. The agent, represented by a red triangle, can rotate left and right, and move forward. Let's say that the set of propositions $\mathcal{P}$ includes R, G, and B, which are true if and only if the agent is standing on a red/green/blue square (respectively) in the current time step.

Then, we can define a wide variety of tasks using LTL:

- Single goal: $\Diamond \mathrm{R}$ (reach a red square).
- Goal sequences: $\Diamond(\mathrm{R} \wedge \Diamond \mathrm{G})$ (reach red and then green).
- Disjunctive goals: $\Diamond \mathrm{R} \vee \Diamond \mathrm{G}$ (reach red or green).
- Conjunctive goals: $\Diamond \mathrm{R} \wedge \Diamond \mathrm{G}$ (reach red and green[1]).
- Safety constraints: $\Box \neg \mathrm{B}$ (do not touch a blue square).

We can also combine these tasks to define new tasks, e.g., "*go to a red square and then a green square but do not touch a blue square*" can be expressed as $\Diamond(\mathrm{R} \wedge \Diamond \mathrm{G}) \wedge \Box \neg \mathrm{B}$.

While LTL is interpreted over infinite sequences, the truth of many LTL formulas can be ensured after a finite number of steps. For instance, the formula $\Diamond \mathrm{R}$ (*eventually red*) is satisfied by any infinite sequence where R is true at some point. Hence, as soon as R holds in a finite sequence, we know that $\Diamond \mathrm{R}$ will hold. Similarly, a formula such as $\Box \neg \mathrm{B}$ is immediately determined to be unsatisfied by an occurrence of B, regardless of what follows.

3.2. From LTL Instructions to Rewards

So far, our discussion about LTL instructions has been environment-agnostic (the syntax and semantics of LTL are independent of the environment). Now, we show how to reward an RL agent for realizing LTL instructions via an MDP. Following previous works (Toro Icarte et al., 2018a; Jothimurugan et al., 2019), we accomplish this by using a *labelling function* $L : S \times A \to 2^{\mathcal{P}}$. The labelling function $L(s, a)$ assigns truth values to the propositions in $\mathcal{P}$ given the current state $s \in S$ of the environment and the action $a \in A$ selected by the agent. One may think of the labelling function as having a collection of *event detectors* that fire when the propositions in $\mathcal{P}$ hold in the environment. In our running example, $\mathrm{R} \in L(s, a)$ iff the agent is on top of the red square and similarly for G (green) and B (blue).

Given a labelling function, the agent can automatically evaluate whether an LTL instruction has been satisfied or falsified. If the instruction is satisfied (i.e., completed) we give the agent a reward of 1 and if the instruction is falsified (e.g., the agent breaks a safety constraint) we penalize the agent with a reward of -1. The episode ends as soon as the instruction is satisfied or falsified. Formally, given an LTL instruction φ over $\mathcal{P}$ and a labelling function $L : S \times A \to 2^{\mathcal{P}}$ and the sequence of states and actions seen so far in the episode: $s_1, a_1, ..., s_t, a_t$, the reward function is defined as follows:

$$R_\varphi(s_1, a_1, ..., s_t, a_t) = \begin{cases} 1 & \text{if } \sigma_1 ... \sigma_t \models \varphi \\ -1 & \text{if } \sigma_1 ... \sigma_t \models \neg\varphi \\ 0 & \text{otherwise} \end{cases}, \quad (1)$$

where $\sigma_i = L(s_i, a_i)$.

[1]In any order.

Observe that the reward function specified above renders a non-zero reward if the LTL formula can be determined to be satisfied or unsatisfied in a finite number of steps. This is guaranteed to be the case for various fragments of LTL, including co-safe LTL (Kupferman & Vardi, 2001) and for so-called LTL-f (the variant of LTL that is interpreted over finite traces). For LTL formulas that cannot be verified or falsified in finite time (e.g. $\Box \Diamond G$), the agent receives no meaningful reward signal. One way to address such LTL formulas is to alter the reward function to render an appropriate reward after a very large but finite number of steps (e.g., 10^6 steps), with commensurate guarantees regarding the resulting policies. The topic of an appropriate reward function for general LTL formulas is addressed in (Hasanbeig et al., 2018) and explored in (Littman et al., 2017).

Finally, note that this reward function might be non-Markovian, as it depends on sequences of states and actions, making the overall learning problem partially observable. We discuss how to deal with this issue below.

3.3. Instructing RL Agents using LTL

We now formalize the problem of learning a policy that can follow LTL instructions.[2] Given an MDP without a reward function $\mathcal{M}_e = \langle S, T, A, \mathbb{P}, \gamma, \mu \rangle$, a finite set of propositional symbols $\mathcal{P}$, a labelling function $L : S \times A \to 2^{\mathcal{P}}$, a finite (but potentially large) set of LTL formulas Φ, and a probability distribution τ over those formulas $\varphi \in \Phi$, our goal is to learn an optimal policy $\pi^*(a_t | s_1, a_1, ..., s_t, \varphi)$ w.r.t. $R_\varphi(s_1, a_1, ..., s_t, a_t)$ for all $\varphi \in \Phi$. To learn this policy, the agent will sample a new LTL task φ from τ on every episode and, during that episode, it will be rewarded according to R_φ. The episode ends when the task is completed, falsified, or a terminal state is reached.

A major challenge to solving this problem is that the optimal policy $\pi^*(a_t | s_1, a_1, ..., s_t, \varphi)$ has to consider the whole history of states and actions since the reward function is non-Markovian. To handle this issue, Kuo et al. (2020) proposed to encode the policy using a recurrent neural network. However, here we show that we can overcome this complexity by exploiting a procedure known as LTL progression (Bacchus & Kabanza, 2000).

Definition 3.1. Given an LTL formula φ and a truth assignment σ over $\mathcal{P}$, $\mathrm{prog}(\sigma, \varphi)$ is defined as follows:

- $\mathrm{prog}(\sigma, p) = \mathsf{true}$ if $p \in \sigma$, where $p \in \mathcal{P}$
- $\mathrm{prog}(\sigma, p) = \mathsf{false}$ if $p \notin \sigma$, where $p \in \mathcal{P}$
- $\mathrm{prog}(\sigma, \neg\varphi) = \neg\, \mathrm{prog}(\sigma, \varphi)$
- $\mathrm{prog}(\sigma, \varphi \wedge \psi) = \mathrm{prog}(\sigma, \varphi) \wedge \mathrm{prog}(\sigma, \psi)$
- $\mathrm{prog}(\sigma, \bigcirc\varphi) = \varphi$
- $\mathrm{prog}(\sigma, \varphi \,\mathsf{U}\, \psi) = \mathrm{prog}(\sigma, \psi) \vee (\mathrm{prog}(\sigma, \varphi) \wedge \varphi \,\mathsf{U}\, \psi)$

[2]Hereafter, LTL instruction/task may be used interchangeably.

The prog operator is a semantics-preserving rewriting procedure that takes an LTL formula and current labelled state as input and returns a formula that identifies aspects of the original instructions that remain to be addressed. Progress towards completion of the task is reflected in diminished remaining instructions. For instance, the task $\Diamond(\mathrm{R} \wedge \Diamond\mathrm{G})$ (*go to red and then to green*) will progress to $\Diamond\mathrm{G}$ (*go to green*) as soon as R holds in the environment. We use LTL progression to make the reward function R_φ Markovian. We achieve this by (1) augmenting the MDP state with the current LTL task φ that the agent is solving, (2) progressing φ after each step given by the agent in the environment, and (3) rewarding the agent when φ progresses to true ($+1$) or false (-1). This gives rise to an augmented MDP, that we call a *Taskable MDP*, where the LTL instructions are part of the MDP states:

Definition 3.2 (Taskable MDP)**.** Given an MDP without a reward function $\mathcal{M}_e = \langle S, T, A, \mathbb{P}, \gamma, \mu\rangle$, a finite set of propositional symbols $\mathcal{P}$, a labelling function $L : S \times A \to 2^{\mathcal{P}}$, a finite set of LTL formulas Φ, and a probability distribution τ over Φ, we construct *Taskable MDP* $\mathcal{M}_\Phi = \langle S', T', A, \mathbb{P}', R', \gamma, \mu'\rangle$, where $S' = S \times \mathrm{cl}(\Phi)$, $T' = \{\langle s, \varphi\rangle \mid s \in T \text{ or } \varphi \in \{\mathsf{true}, \mathsf{false}\}\}$, $\mathbb{P}'(\langle s', \varphi'\rangle | \langle s, \varphi\rangle, a) = \mathbb{P}(s'|s, a)$ if $\varphi' = \mathrm{prog}(L(s, a), \varphi)$ (zero otherwise), $\mu'(\langle s, \varphi\rangle) = \mu(s) \cdot \tau(\varphi)$, and

$$R'(\langle s, \varphi\rangle, a) = \begin{cases} 1 & \text{if } \mathrm{prog}(L(s,a), \varphi) = \mathsf{true} \\ -1 & \text{if } \mathrm{prog}(L(s,a), \varphi) = \mathsf{false} \\ 0 & \text{otherwise} \end{cases}.$$

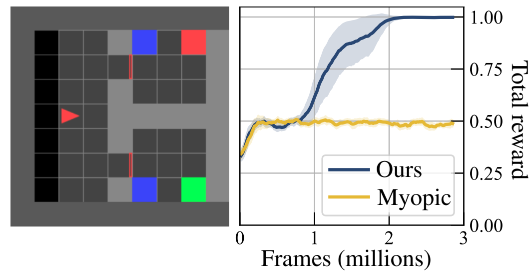
Here $\mathrm{cl}(\Phi)$ denotes the *progression closure* of Φ, i.e., the smallest set containing Φ that is closed under progression.

With that, the main theorem of our paper shows that an optimal policy $\pi^*(a_t|s_1, a_1, ..., s_t, \varphi)$ to solve any LTL task $\varphi \in \Phi$ in some environment $\mathcal{M}_e$ achieves the same expected discounted return as an optimal policy $\pi^*(a|s, \varphi)$ for the Taskable MDP $\mathcal{M}_\Phi$ constructed using $\mathcal{M}_e$ and Φ (we prove this theorem in Appendix A).

Theorem 3.1. *Let $\mathcal{M}_\Phi = \langle S', T', A, \mathbb{P}', R', \gamma, \mu'\rangle$ be a Taskable MDP constructed from an MDP without a reward function $\mathcal{M}_e = \langle S, T, A, \mathbb{P}, \gamma, \mu\rangle$, a finite set of propositional symbols $\mathcal{P}$, a labelling function $L : S \times A \to 2^{\mathcal{P}}$, a finite set of LTL formulas Φ, and a probability distribution τ over Φ. Then, an optimal stationary policy $\pi^*_\Phi(a|s, \varphi)$ for $\mathcal{M}_\Phi$ achieves the same expected discounted return as an optimal non-stationary policy $\pi^*_\varphi(a_t|s, a_1, ..., s_t, \varphi)$ for $\mathcal{M}_e$ w.r.t. R_φ, as defined in (1), for all $s \in S$ and $\varphi \in \Phi$.*

3.4. Discussion and Bibliographical Remarks

Two recent works have explored how to teach RL agents to follow unseen instructions using temporal logic (Kuo et al., 2020; Leon et al., 2020). Here we discuss the theoretical



(a) (b)

Figure 1. (a) A toy minigrid environment where doors lock upon entry. The task is equally likely to be either *go to blue then red* or *go to blue then green*. (b) A Myopic policy only succeeds in 50% of tasks while our approach obtains the maximum reward.

advantages of our approach over theirs. Kuo et al. propose to learn a policy $\pi^*(a_t|s_1, a_1, ..., s_t, \varphi)$ (using a recurrent neural network) by solving a partially observable problem (i.e., a POMDP). In contrast, we propose to learn a policy $\pi^*(a_t|s_t, \varphi)$ in a Taskable MDP $\mathcal{M}_\Phi$. Since solving MDPs is easier than solving POMDPs (MDPs can be solved in polynomial time whereas POMDPs are undecidable), this gives our approach a theoretical advantage which results in better empirical performance (as shown in Section 5).

Leon et al. (2020) follow a different approach. They instruct agents using a fragment of LTL (which only supports the temporal operator *eventually*) and define a reasoning module that automatically returns a proposition to satisfy which makes progress towards solving the task. Thus, the agent only needs to learn a policy $\pi(a|s, p)$ conditioned on the state s and a proposition $p \in \mathcal{P}$. However, this approach is myopic – it optimizes for solving the next subtask without considering what the agent must do after and, as a result, might converge to suboptimal solutions. This is a common weakness across recent approaches that instruct RL agents (e.g., Sohn et al., 2018; Jiang et al., 2019; Sun et al., 2019).

As an example, consider (again) the MiniGrid from Figure 1. Observe the two red doors at the entrance to each room. These doors automatically lock upon entry so the agent cannot visit both rooms. Suppose the agent has to solve two LTL tasks, uniformly sampled at the beginning of each episode: $\Diamond(\mathrm{B} \wedge \Diamond\mathrm{G})$ (*go to a blue square and then to a green square*) or $\Diamond(\mathrm{B} \wedge \Diamond\mathrm{R})$ (*go to a blue square and then to a red square*). For both tasks, a myopic approach will tell the agent to first achieve $\Diamond\mathrm{B}$ (*go to blue*), but doing so without considering where the agent must go after might lead to a dead end (due to the locking doors). In contrast, an approach that learns an optimal policy $\pi^*(a|s, \varphi)$ for $\mathcal{M}_\Phi$ can consistently solve these two tasks (Figure 1(b)).

The theoretical advantages of our approach however comes with a cost. Learning $\pi^*(a|s, \varphi)$ is harder than learning a myopic policy $\pi^*(a|p)$ and, hence, it seems reasonable to expect that a myopic approach will generalize better to unseen instructions. However, we did not observe this behaviour in our experiments.

4. Model Architecture

In this section, we build on the RL framework with LTL instructions from Section 3 and explain a way to realize this approach in complex environments using deep RL.

In each episode, a new LTL task φ is sampled. At every step, the environment returns an observation, the event detectors return a truth assignment σ of all propositions in $\mathcal{P}$ and the LTL task is automatically progressed to $\varphi := \text{prog}(\sigma, \varphi)$. We used `Spot` (Duret-Lutz et al., 2016) to simplify φ to an equivalent form after each progression. The agent then receives both the environment observation and the progressed formula and emits an action via a modular architecture consisting of three trainable components (see Figure 2(a)):

1. Env Module: an environment-dependent model that preprocesses the observations (e.g., a convolutional or a fully-connected network).

2. LTL Module: a neural encoder for the LTL instructions (discussed below).

3. RL Module: a module which decides actions to take in the environment, based on observations encoded by the Env Module and the current (progressed) task encoded by the LTL module. While our approach is agnostic to the choice of RL algorithm, in our experiments we opted for *Proximal Policy Optimization* (PPO) (Schulman et al., 2017) for its strong generalization performance (Cobbe et al., 2019).

4.1. LTL Module

LTL formulas can be encoded through different means, the simplest of which is to apply a sequence model (e.g., LSTM) to the input formula. However, given the tree-structured nature of these formulas, *Graph Neural Networks* (GNNs) (Gori et al., 2005; Scarselli et al., 2008) may provide a better inductive bias. This selection is in line with recent works in programming languages and verification, where GNNs are used to embed the *abstract syntax tree* (AST) of the input program (Allamanis et al., 2018; Si et al., 2018).

Out of many incarnations of GNNs, we choose a version of *Relational Graph Convolutional Network* (R-GCN) (Schlichtkrull et al., 2018). R-GCN works on labeled graphs $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ with nodes $v, u \in \mathcal{V}$ and typed edges $(v, r, u) \in \mathcal{E}$, where $r \in \mathcal{R}$ is an edge type. Given G and a set of input node features $\{\boldsymbol{x}_v^{(0)} | \forall v \in \mathcal{V}\}$, the R-GCN maps the nodes to a vector space, through a series of *message passing* steps. At step t, the embedding $\boldsymbol{x}_v^{(t)} \in \mathbb{R}^{d^{(t)}}$ of node v is updated by a normalized sum of the transformed feature vectors of neighbouring nodes, followed by an element-wise activation function $\sigma(.)$:

$$\boldsymbol{x}_v^{(t+1)} = \sigma\left(\sum_{r \in \mathcal{R}} \sum_{u \in \mathcal{N}_G^r(v)} \frac{1}{|\mathcal{N}_G^r(v)|} W_r \boldsymbol{x}_u^{(t)} \right), \quad (2)$$

where $\mathcal{N}_G^r(v)$ denotes the set of nodes adjacent to v via an edge of type $r \in \mathcal{R}$. Note that different edge types use different weights W_r and weight-sharing is done only for edges of the same type at each iteration.

We represent an LTL formula φ as a directed graph $G_\varphi = (V_\varphi, E_\varphi, R)$, as shown in Figure 2(b). This is done by first creating φ's parse tree, where each subformula is connected to its parent operator via a directed edge, and then adding self-loops for all the nodes. We distinguish between $|\mathcal{R}| = 4$ edge types: 1. *Self-loops*, 2. *Unary*: for connecting the subformula of a unary operator to its parent node, 3. *Binary_left* (*resp.* 4. *Binary_right*): for connecting the left (*resp.* right) subformula of a binary operator to its parent node. R-GCN performs T message passing steps according to Equation (2) over G_φ. The inclusion of self-loops is to ensure that the representation of a node at step $t+1$ is also informed by its corresponding representation at step t. Due to the direction of the edges in G_φ, the messages flow in a bottom-up manner and after T steps we regard the embedding of the root node of G_φ as the embedding of φ.

We experimented with both sequence models and R-GCN. In each case we first create one-hot encodings of the formula tokens (operators and propositions). For sequence models we convert the formula to its prefix notation $pre(\varphi)$ and then replace the tokens with their one-hot encodings. For R-GCN, these encodings serve as input node features $\boldsymbol{x}_v^{(0)}$.

4.2. Pretraining the LTL Module

Simultaneous training of the LTL Module with the rest of the model can be a strenuous task. We propose to pretrain the LTL module, taking advantage of the environment-agnostic nature of LTL semantics and the agent's modular architecture. Formally, given a target Taskable MDP $\mathcal{M}_\Phi$, our aim is to learn useful encodings for formulas in Φ and later use those encodings in solving $\mathcal{M}_\Phi$. We cast the pretraining itself as solving a special kind of Taskable MDP.

Definition 4.1 (`LTLBootcamp`)**.** Given a set of formulas Φ and a distribution τ over Φ, we construct a single-state MDP (without the reward function) $\mathcal{M}_\varnothing = \langle S, T, A, \mathbb{P}, \gamma, \mu \rangle$, where $S = \{s_0\}$, $T = \varnothing$, $A = \mathcal{P}$, $\mathbb{P}(s_0 | s_0, .) = 1$, and $\mu(s_0) = 1$. The labelling function is given by $L(s_0, p) = \{p\}$. Finally, the `LTLBootcamp` environment is defined to be the Taskable MDP given by $\mathcal{M}_\varnothing$, $\mathcal{P}$, L, Φ, and τ.

Intuitively, the `LTLBootcamp` task is to progress formulas $\varphi \sim \tau(\Phi)$ to true in as few steps as possible by setting a

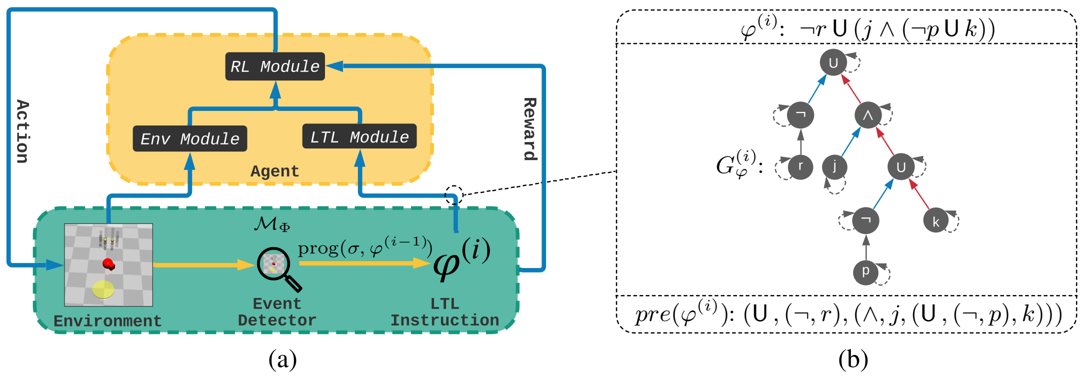


Figure 2. (a) Our RL framework for solving LTL tasks with the Taskable MDP $\mathcal{M}_\Phi$ and our agent's modular structure. In each step, the agent receives an environment observation and an LTL instruction as input. LTL instructions are automatically progressed based on signals from the event detectors, allowing a Markov policy to solve non-Markovian tasks. (b) Possible representations of an LTL formula. *Top*: standard LTL syntax. *Middle*: abstract syntax tree. *Bottom*: prefix notation.

single proposition to true at each step. Hence our scheme is: (1) train to convergence on `LTLBootcamp` with formula set Φ and task distribution τ of the target Taskable MDP $\mathcal{M}_\Phi$; (2) transfer the learned LTL Module as the initial LTL Module in $\mathcal{M}_\Phi$. While many pretraining schemes are possible, this one involves a simple task which can be viewed as an abstracted version of the downstream task.

Since pretraining does not require interaction with a physical environment, it is more wall-clock efficient than training the full model on the downstream environment. Furthermore, the LTL Module is robust to changes in the environment as long as Φ and τ remain the same, thanks to the modular architecture of our model. In Section 5 we demonstrate the empirical benefits of pretraining the LTL Module.

5. Experiments

We designed our experiments to investigate whether RL agents can learn to solve complex, temporally extended tasks specified in LTL. Specifically we answer the following questions: **(1) Performance:** How does our approach fare against baselines that do not utilize LTL progression or are myopic? **(2) Architecture:** What's the effect of different architectural choices on our model's performance? **(3) Pretraining:** Does pretraining the LTL Module result in more rapid convergence in novel downstream environments? **(4) Upward Generalization:** Can the RL agent trained using our approach generalize to larger instructions than those seen in training? **(5) Continuous Action-Space:** How does our approach perform in a continuous action-space domain?

Our code and videos of our agents are available at `github.com/LTL2Action/LTL2Action`.

5.1. Experimental Setup

We ran experiments across different environments and LTL tasks, where the tasks vary in length and difficulty to form an implicit curriculum. To measure how well each approach generalizes to unseen instructions we followed the methodology proposed by Cobbe et al. (2019). In every episode, the agent faces some LTL task sampled i.i.d. from a large set of possible tasks Φ. We evaluate how well the learned policy generalizes to new samples from Φ, the majority of which have not been previously seen, with high probability. We also consider out-of-distribution generalization to larger tasks than those in Φ, which are guaranteed to be unseen.

5.1.1. Environments

We use the following environments in our experiments:

`LetterWorld`: A 7×7 discrete grid environment, similar to Andreas et al. 2017. Out of the 49 squares, 24 are associated with 12 unique propositions/letters (each letter appears twice in the grid, allowing more than one way to satisfy any proposition). At each step the agent can move along the cardinal directions. The agent observes the full grid (and letters) from an egocentric point of view as well as the current LTL task (γ=0.94, timeout=75 steps).

`ZoneEnv`: We co-opted OpenAI's Safety Gym (Ray et al., 2019) which has a continuous action-space. Our environment (Figure 5(a)) is a walled 2D plane with 8 circles (2 of each colour), called "zones," that correspond to task propositions. We use Safety Gym's Point robot with actions for steering and forward/backward acceleration. It observes lidar information towards the zones and other sensory data (e.g., accelerometer, velocimeter). The zones and the robot are randomly positioned on the plane at the start of each episode and the robot has to visit and/or avoid certain zones based on the LTL task (γ=0.998, timeout=1000 steps).

`LTLBootcamp`: The Taskable MDP from Section 4, only used to pretrain the LTL Module (γ=0.9, timeout=75 steps).

5.1.2. TASKS

Our experiments consider two LTL task spaces, where tasks are randomly sampled via procedural generation. We provide a high-level description of the two task spaces, with more details in Appendix B.1.

Partially-Ordered Tasks: A task consists of multiple sequences of propositions which can be solved in parallel. However, the propositions within each sequence must be satisfied in order. For example, a possible task (specified informally in English) is: *"satisfy C, A, B in that order, and satisfy D, A in that order"* – where one valid solution would be to satisfy D, C, A, B in that order. The number of possible unique tasks is over 5×10^{39}.

Avoidance Tasks: This set of tasks is similar to Partially-Ordered Tasks, but includes propositions that must also be avoided (or else the task is failed). The propositions to avoid change as different parts of the task are solved. The number of possible unique tasks is over 970 million.

Note that the formulas we consider contain up to 75 tokens (propositions and operators) in training and 210 tokens in the upward generalization experiments, while past related works only considered up to 20 tokens (Kuo et al., 2020; Leon et al., 2020).

5.1.3. OUR METHODS AND BASELINES

We experimented with three variants of our approach, all exploiting LTL progression and utilizing PPO for policy optimization. They differed in the type of LTL Module used, namely: GNN, GRU, and LSTM. In our plots, we refer to these approaches as GNN_{prog}, GRU_{prog}, and, $\text{LSTM}_{\text{prog}}$, respectively. Details about neural network architectures and PPO hyperparameters can be found in Appendix Sections B.2, B.3, respectively.

We compared our method against three baselines. The *No LTL* baseline ignores the LTL instructions, but learns a non-stationary policy using an LSTM. This baseline tells us if the agent can learn a policy that works well regardless of the LTL instruction. The *GRU* baseline is inspired by Kuo et al. (2020). This approach learns a policy that considers the LTL instructions but does not progress the formula over time. Instead, it learns a non-stationary policy encoded using a GRU (as discussed in Section 3.4).

Lastly, the *Myopic* baseline was inspired by Leon et al. (2020) and other similar approaches (e.g., Andreas et al., 2017; Oh et al., 2017; Xu et al., 2018; Sohn et al., 2018; Sun et al., 2019). In this baseline, a reasoning technique

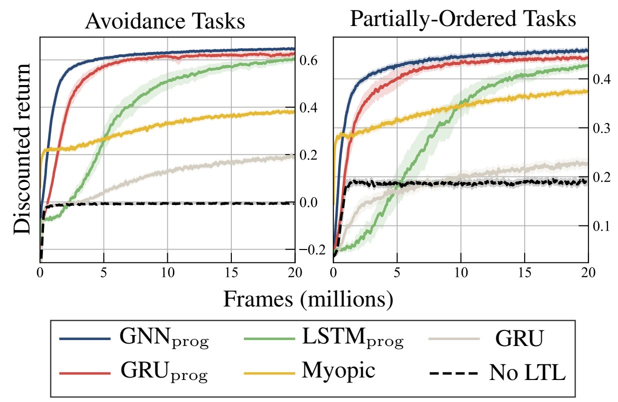


Figure 3. Our approaches using LTL progression (marked by $\bullet_{\text{prog}}$) outperformed other baselines on `LetterWorld`. We report *discounted return* over the duration of training (averaged over 30 seeds, with 90% confidence intervals).

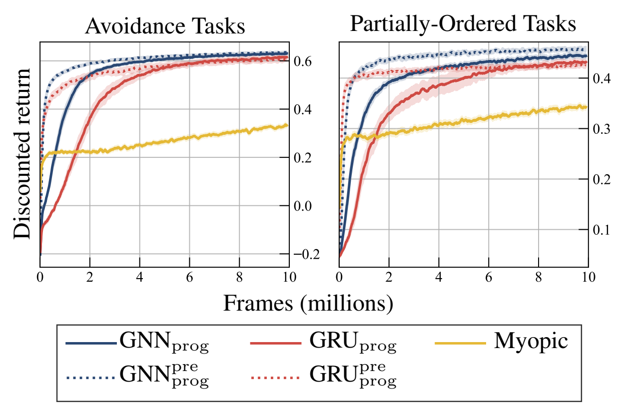


Figure 4. Pretrained LTL models (marked by $\bullet^{\text{pre}}$) showed better sample-efficiency than non-pretrained versions on `LetterWorld`. The *Myopic* baseline is shown for comparison. We report *discounted return* over the duration of training (averaged over 30 seeds, with 90% confidence intervals).

is used to tell the agent which propositions to achieve next in order to solve the LTL task. Specifically, the agent observes whether making a particular proposition true would (a) progress the current formula, (b) have no effect, or (c) make it unsatisfiable. Given these observations, the agent then learns a Markovian policy. Note that this approach might converge to suboptimal solutions (see Section 3.4).

5.2. Results

We conduct our experiments on `LetterWorld`, except for the continuous action-space tests where we used `ZoneEnv`.

Performance Figure 3 shows the results on Partially-Ordered and Avoidance Tasks when tested on i.i.d. samples from Φ. The results show that: (1) LTL progression significantly improved generalization, (2) A compositional architecture such as GNN learned to encode LTL formulas

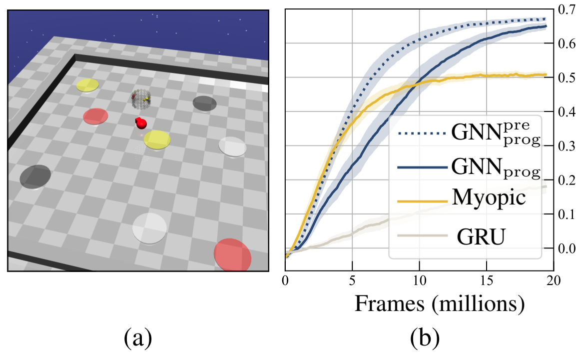


Figure 5. (a) The `ZoneEnv` continuous control environment with coloured zones as LTL propositions. Tasks involve reaching zones of certain colours in the correct order (while avoiding zones of the incorrect colour in the Avoidance Task). (b) Both GNN architectures outperformed the *Myopic* and GRU without progression baselines on `ZoneEnv` on Avoidance Tasks. Pretraining resulted in faster convergence. We report *discounted return* over the duration of training (averaged over 30 seeds, with 90% confidence intervals).

better than sequential ones such as GRU and LSTM, and (3) The myopic baseline initially learned quickly, but it was eventually outperformed by all our methods.

Pretraining We investigated the effects of pretraining the LTL Module (Section 4.2). Figure 4 shows the results for GNN and GRU encoders, with and without pretraining, as well as the myopic baseline. We observed that pretraining the LTL Module accelerated learning for both encoders.

Upward Generalization The preceding experiments indicate that an RL agent trained with our approach generalizes to new, incoming tasks from a large, but fixed training distribution Φ. Here, we consider *upward generalization* to larger tasks than seen in training.

We evaluated trained agents on Partially-Ordered and Avoidance Tasks with: (a) longer sequences, and (b) more conjuncts (i.e., more tasks to be completed in parallel) than seen in training. For Avoidance Tasks, we increased the max depth of formulas from 3 (in training) to 6, and the max number of conjuncts from 2 to 3. For Partially-Ordered Tasks, we increased the max depth from 5 to 15, and the max number of conjuncts from 4 to 12.

We report the generalization performance of various baselines in Table 1. The myopic and LTL progression-based approaches significantly outperformed the GRU baseline without progression, suggesting that decomposing the task is essential for generalization. Pretraining also marginally improved the GNN (with progression) baseline. We highlight the impact of architecture on generalization – GNN outperformed GRU in most cases. This aligns with other works showing scalability of GNNs to larger formulas (Selsam et al., 2018; Vaezipoor et al., 2020). Note that upward

Table 1. Trained RL agents are evaluated on the training distribution of tasks, as well as out-of-distribution tasks with increased depth of sequences, and increased number of conjuncts. In each entry, we report *total reward* and *discounted return* (in parentheses) averaged over 30 seeds and 100 episodes per seed. The highest value over all approaches is bolded.

	I.I.D	↑ Depth	↑ Conjuncts
	(a) Avoidance Tasks		
GNN$^{pre}_{prog}$	0.97(0.65)	**0.99**(**0.35**)	**0.64**(0.18)
GNN$_{prog}$	**0.98**(**0.66**)	0.98(0.33)	0.57(0.14)
GRU$_{prog}$	0.89(0.63)	0.42(0.07)	0.58(**0.25**)
GRU	0.32(0.22)	-0.03(-0.01)	-0.35(-0.16)
Myopic	0.88(0.50)	0.71(0.07)	0.58(0.11)
	(b) Partially-Ordered Tasks		
GNN$^{pre}_{prog}$	**1.0**(**0.48**)	**0.98**(**0.0088**)	**0.99**(**0.0380**)
GNN$_{prog}$	**1.0**(0.47)	0.97(0.0074)	0.98(0.0340)
GRU$_{prog}$	**1.0**(0.46)	0.29(0.0005)	**0.99**(0.0252)
GRU	0.87(0.24)	0.03(0.0000)	0.54(0.0075)
Myopic	**1.0**(0.40)	0.94(0.0042)	**0.99**(0.0221)

generalization on conjuncts for Avoidance tasks is particularly challenging since only up to 2 conjuncts were observed in training.

Continuous Action-Space Figure 5 shows the results on `ZoneEnv` for a reduced version of the Avoidance Task. We note that our approaches (GNN$^{pre}_{prog}$ and GNN$_{prog}$) solved almost all tasks, however *Myopic* and the GRU baseline without progression failed to solve many tasks within the timeout. These results reaffirm the generalizability of our approach on a continuous environment.

6. Discussion

Our experiments demonstrated the following key findings: **(a)** encoding full task instructions converges to better solutions than myopic methods; **(b)** LTL progression improves learning and generalization; **(c)** LTL semantics can be pretrained to improve downstream learning; **(d)** our method can zero-shot generalize to new instructions significantly larger than those seen in training.

We note that architecture is an important factor for encoding LTL. GNNs appear to more effectively encode the compositional syntax of LTL, whereas, GRUs are more wall-clock efficient (by roughly 2 to 3×) due to the overhead of constructing abstract syntax trees for GNNs.

Our results are encouraging and open several directions for future work. This includes exploring ways to build general LTL models which fully capture LTL semantics without assuming access to a distribution of tasks. Similar to most works focusing on formal language in RL (e.g. Toro Icarte et al. 2018a; Jothimurugan et al. 2019; Leon et al. 2020), we assume a noise-free *labelling function* is available to identify high-level domain features. An important question is whether this labelling function can be learned, and how an RL agent can handle the resultant uncertainty.

In this work, we investigated generalization to new formulas over a fixed set of propositions. However, it is also interesting to study how to generalize to formulas with *new propositions*. We note that some existing works have tackled this setting (Hill et al., 2021; Leon et al., 2020; Lake, 2019). One way of extending our framework to also generalize to *unseen* propositions is to encode the propositions using some feature representation other than a one-hot encoding. We include some preliminary experiments in Appendix C.1 showing that, by changing the feature representation of the propositional symbols, our framework is indeed able to generalize to tasks with unseen objects and propositions. But further investigation is needed.

7. Related Work

This paper builds on past work in RL which explores using LTL (or similar formal languages) for reward function specification, decomposition, or shaping (e.g., Aksaray et al., 2016; Li et al., 2017; Littman et al., 2017; Toro Icarte et al., 2018b; Li et al., 2018; Camacho et al., 2017; 2019; Yuan et al., 2019; Jothimurugan et al., 2019; Xu & Topcu, 2019; Hasanbeig et al., 2018; 2020; de Giacomo et al., 2020a;b; Jiang et al., 2020). However, most of these methods are limited to learning a single, fixed task in LTL. Toro Icarte et al. (2018a) explicitly focuses on learning multiple LTL tasks optimally, but their approach is unable to generalize to unseen tasks and may have to learn an exponential number of policies in the length of the largest formula.

In this work, we consider a multitask setting in which a new task is sampled each episode from a large task space. Our motivation is to enable an agent to solve unseen tasks without further training, similar in spirit to previous works (e.g. Andreas et al. 2017; Xu et al. 2018; Oh et al. 2017; Sohn et al. 2018), some of which also considers temporal logic tasks (Leon et al. 2020; Kuo et al. 2020). A common theme in all the previous listed works (except for one: Kuo et al. 2020) is to decompose large tasks into independent, sequential subtasks, however this often performs suboptimally in solving the full task. We instead consider the full LTL task, as also adopted by Kuo et al. 2020. We additionally propose to use LTL progression to enable standard, Markovian learning – drastically improving both sample and wall-clock efficiency – and pretraining the LTL module to accelerate learning. Note that Kuo et al. 2020 do not encode LTL formulas, but instead compose neural networks to mirror the formula structure, which is incompatible with LTL pretraining as it is environment-dependent.

Other representations of task specifications for RL have also been proposed, including programs (Sun et al., 2019; Fasel et al., 2009; Denil et al., 2017), policy sketches (Andreas et al., 2017), and natural language (Jiang et al. 2019; Bahdanau et al. 2018; see Luketina et al. 2019 for an extensive survey). Finally, note that it is possible to automatically translate natural language instructions into LTL (e.g., Dzifcak et al., 2009; Brunello et al., 2019; Wang et al., 2020).

8. Conclusion

Creating learning agents that understand and follow open-ended human instructions is a challenging problem with significant real-world applicability. Part of the difficulty stems from the need for large training sets of instructions and associated rewards. In this work, we trained an RL agent to follow temporally extended instructions specified in the formal language, LTL. The compositional syntax of LTL allowed us to generate massive training data, automatically. We theoretically motivate a novel approach to multitask RL using neural encodings of LTL instructions and LTL progression. Our experiments on discrete and continuous domains demonstrated robust generalization across procedurally generated task sets, outperforming a prevalent myopic approach.

LTL is a popular specification language for synthesis and verification, and is used for robot tasking and human-robot interaction (e.g., Dzifcak et al., 2009; Raman et al., 2012; Li et al., 2017; Moarref & Kress-Gazit, 2017; 2020; Kasenberg & Scheutz, 2017; Shah et al., 2018; 2020). We believe the contributions in this paper have the potential for broad impact within these communities, and could be adapted for other structured languages.

Acknowledgements

We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canada CIFAR AI Chairs Program, and Microsoft Research. The third author also gratefully acknowledges funding from ANID (Becas Chile). Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute for Artificial Intelligence www.vectorinstitute.ai/partners. We thank the Schwartz Reisman Institute for Technology and Society for providing a rich multi-disciplinary research environment. Additionally, we thank our anonymous reviewers for their feedback which led to several improvements in this paper.

References

Aksaray, D., Jones, A., Kong, Z., Schwager, M., and Belta, C. Q-learning for Robust Satisfaction of Signal Temporal

Logic Specifications. In *Proceedings of the 55th IEEE Annual Conference on Decision and Control (CDC)*, pp. 6565–6570. IEEE, 2016.

Allamanis, M., Brockschmidt, M., and Khademi, M. Learning to Represent Programs with Graphs. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

Andreas, J., Klein, D., and Levine, S. Modular Multitask Reinforcement Learning with Policy Sketches. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 166–175. PMLR, 2017.

Bacchus, F. and Kabanza, F. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.

Bahdanau, D., Hill, F., Leike, J., Hughes, E., Hosseini, A., Kohli, P., and Grefenstette, E. Learning to Understand Goal Specifications by Modelling Reward. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

Baier, C. and Katoen, J. *Principles of Model Checking*. MIT Press, 2008.

Brunello, A., Montanari, A., and Reynolds, M. Synthesis of LTL formulas from natural language texts: State of the art and research directions. In *Proceedings of the 26th International Symposium on Temporal Representation and Reasoning (TIME)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

Camacho, A., Chen, O., Sanner, S., and McIlraith, S. A. Decision-Making with non-Markovian Rewards: From LTL to Automata-based Reward Shaping. In *Proceedings of the 3rd Multi-disciplinary Conference on Reinforcement Learning and Decision (RLDM)*, pp. 279–283, 2017.

Camacho, A., Toro Icarte, R., Klassen, T. Q., Valenzano, R., and McIlraith, S. A. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 19, pp. 6065–6073, 2019.

Chaplot, D. S., Sathyendra, K. M., Pasumarthi, R. K., Rajagopal, D., and Salakhutdinov, R. Gated-Attention Architectures for Task-Oriented Language Grounding. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, volume 32, 2018.

Chevalier-Boisvert, M., Willems, L., and Pal, S. Minimalistic Gridworld Environment for OpenAI Gym. https://github.com/maximecb/gym-minigrid, 2018.

Co-Reyes, J. D., Gupta, A., Sanjeev, S., Altieri, N., DeNero, J., Abbeel, P., and Levine, S. Meta-Learning Language-Guided Policy Learning. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. Quantifying Generalization in Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 1282–1289. PMLR, 2019.

de Giacomo, G., Favorito, M., Iocchi, L., Patrizi, F., and Ronca, A. Temporal Logic Monitoring Rewards via Transducers. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, volume 17, pp. 860–870, 2020a.

de Giacomo, G., Iocchi, L., Favorito, M., and Patrizi, F. Restraining Bolts for Reinforcement Learning Agents. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pp. 13659–13662, 2020b.

Denil, M., Colmenarejo, S. G., Cabi, S., Saxton, D., and de Freitas, N. Programmable Agents. *arXiv preprint arXiv:1706.06383*, 2017.

Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., and Xu, L. Spot 2.0—A Framework for LTL and ω-Automata Manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, pp. 122–129. Springer, 2016.

Dzifcak, J., Scheutz, M., Baral, C., and Schermerhorn, P. What to Do and How to Do It: Translating Natural Language Directives Into Temporal and Dynamic Logic Representation for Goal Management and Action Execution. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4163–4168. IEEE, 2009.

Fasel, I. R., Quinlan, M., and Stone, P. A Task Specification Language for Bootstrap Learning. In *Proceedings of the AAAI Spring Symposium on Agents that Learn from Human Teachers*, pp. 48–55, 2009.

Gori, M., Monfardini, G., and Scarselli, F. A New Model for Learning in Graph Domains. In *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks (IJCNN)*, volume 2, pp. 729–734, 2005.

Hasanbeig, M., Abate, A., and Kroening, D. Logically-Constrained Reinforcement Learning. *arXiv preprint arXiv:1801.08099*, 2018.

Hasanbeig, M., Kroening, D., and Abate, A. Deep Reinforcement Learning with Temporal Logics. In *Proceedings of the 18th International Conference on Formal*

Modeling and Analysis of Timed Systems (FORMATS), pp. 1–22. Springer, 2020.

Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W. M., Jaderberg, M., Teplyashin, D., et al. Grounded Language Learning in a Simulated 3D World. *arXiv preprint arXiv:1706.06551*, 2017.

Hill, F., Tieleman, O., von Glehn, T., Wong, N., Merzic, H., and Clark, S. Grounded Language Learning Fast and Slow. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2021.

Jiang, Y., Gu, S., Murphy, K., and Finn, C. Language as an Abstraction for Hierarchical Deep Reinforcement Learning. In *Proceedings of the 32nd Conference on Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pp. 9414–9426, 2019.

Jiang, Y., Bharadwaj, S., Wu, B., Shah, R., Topcu, U., and Stone, P. Temporal-Logic-Based Reward Shaping for Continuing Learning Tasks. *arXiv preprint arXiv:2007.01498*, 2020.

Jothimurugan, K., Alur, R., and Bastani, O. A Composable Specification Language for Reinforcement Learning Tasks. In *Proceedings of the 32nd Conference on Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pp. 13041–13051, 2019.

Kasenberg, D. and Scheutz, M. Interpretable apprenticeship learning with temporal logic specifications. In *Proceedings of the 56th IEEE Annual Conference on Decision and Control (CDC)*, pp. 4914–4921. IEEE, 2017.

Kuo, Y.-L., Katz, B., and Barbu, A. Encoding Formulas as Deep Networks: Reinforcement Learning for Zero-Shot Execution of LTL Formulas. *arXiv preprint arXiv:2006.01110*, 2020.

Kupferman, O. and Vardi, M. Y. Model checking of safety properties. *Formal Methods in System Design*, 19(3): 291–314, 2001.

Lake, B. M. Compositional generalization through meta sequence-to-sequence learning. In *Proceedings of the 32nd Conference on Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Leon, B. G., Shanahan, M., and Belardinelli, F. Systematic Generalisation through Task Temporal Logic and Deep Reinforcement Learning. *arXiv preprint arXiv:2006.08767*, 2020.

Li, X., Vasile, C., and Belta, C. Reinforcement Learning with Temporal Logic Rewards. In *Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3834–3839, 2017.

Li, X., Ma, Y., and Belta, C. A Policy Search Method for Temporal Logic Specified Reinforcement Learning Tasks. In *Proceedings of the 2018 Annual American Control Conference (ACC)*, pp. 240–245. IEEE, 2018.

Littman, M. L., Topcu, U., Fu, J., Isbell, C., Wen, M., and MacGlashan, J. Environment-Independent Task Specifications via GLTL. *arXiv preprint arXiv:1704.04341*, 2017.

Luketina, J., Nardelli, N., Farquhar, G., Foerster, J., Andreas, J., Grefenstette, E., Whiteson, S., and Rocktäschel, T. A Survey of Reinforcement Learning Informed by Natural Language. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 57, pp. 6309–6317, 2019.

McCarthy, J. et al. *Programs with common sense*. RLE and MIT computation center, 1960.

Moarref, S. and Kress-Gazit, H. Decentralized control of robotic swarms from high-level temporal logic specifications. In *Proceedings of the 2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pp. 17–23. IEEE, 2017.

Moarref, S. and Kress-Gazit, H. Automated synthesis of decentralized controllers for robot swarmsfrom high-level temporal logic specifications. *Autonomous Robots*, 44 (3-4):585–600, 2020.

Oh, J., Singh, S., Lee, H., and Kohli, P. Zero-Shot Task Generalization with Multi-Task Deep Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 2661–2670. PMLR, 2017.

Pnueli, A. The Temporal Logic of Programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 46–57. IEEE, 1977.

Raman, V., Finucane, C., and Kress-Gazit, H. Temporal logic robot mission planning for slow and fast actions. In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 251–256. IEEE, 2012.

Ray, A., Achiam, J., and Amodei, D. Benchmarking Safe Exploration in Deep Reinforcement Learning. *arXiv preprint arXiv:1910.01708*, 2019.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. Modeling Relational Data with Graph Convolutional Networks. In *Proceedings of the*

15th European Semantic Web Conference (ESWC), pp. 593–607. Springer, 2018.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., and Dill, D. L. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.

Shah, A., Kamath, P., Shah, J. A., and Li, S. Bayesian Inference of Temporal Task Specifications from Demonstrations. In *Proceedings of the 31st Conference on Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3808–3817, 2018.

Shah, A., Li, S., and Shah, J. Planning With Uncertain Specifications (PUnS). *IEEE Robotics Autom. Lett.*, 5(2): 3414–3421, 2020.

Si, X., Dai, H., Raghothaman, M., Naik, M., and Song, L. Learning Loop Invariants for Program Verification. In *Proceedings of the 31st Conference on Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Sohn, S., Oh, J., and Lee, H. Hierarchical Reinforcement Learning for Zero-shot Generalization with Subtask Dependencies. In *Proceedings of the 31st Conference on Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pp. 7156–7166, 2018.

Sun, S.-H., Wu, T.-L., and Lim, J. J. Program Guided Agent. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

Toro Icarte, R., Klassen, T. Q., Valenzano, R., and McIlraith, S. A. Teaching Multiple Tasks to an RL Agent using LTL. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 452–461, 2018a.

Toro Icarte, R., Klassen, T. Q., Valenzano, R., and McIlraith, S. A. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 2107–2116, 2018b.

Toro Icarte, R., Klassen, T. Q., Valenzano, R., and McIlraith, S. A. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. *arXiv preprint arXiv:2010.03950*, 2020.

Vaezipoor, P., Lederman, G., Wu, Y., Maddison, C. J., Grosse, R., Lee, E., Seshia, S. A., and Bacchus, F. Learning branching heuristics for propositional model counting. *arXiv preprint arXiv:2007.03204*, 2020.

Wang, C., Ross, C., Katz, B., and Barbu, A. Learning a Natural-Language to LTL Executable Semantic Parser for Grounded Robotics. *arXiv preprint arXiv:2008.03277*, 2020.

Xu, D., Nair, S., Zhu, Y., Gao, J., Garg, A., Fei-Fei, L., and Savarese, S. Neural Task Programming: Learning to Generalize Across Hierarchical Tasks. In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3795–3802. IEEE, 2018.

Xu, Z. and Topcu, U. Transfer of Temporal Logic Formulas in Reinforcement Learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4010–4018, 2019.

Yu, H., Zhang, H., and Xu, W. Interactive Grounded Language Acquisition and Generalization in a 2D World. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

Yuan, L. Z., Hasanbeig, M., Abate, A., and Kroening, D. Modular Deep Reinforcement Learning with Temporal Logic Specifications. *arXiv preprint arXiv:1909.11591*, 2019.