LTL2Action: Generalizing LTL Instructions for Multi-Task RL (Appendix)

A. Proof of Theorem 3.1

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 according to Definition 3.2. 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 for all $s \in S$ and $\varphi \in \Phi$.*

To prove Theorem 3.1, we use a theorem from Bacchus & Kabanza (2000), which shows the correctness of LTL progression:

Theorem A.1. *Given any LTL formula φ and an infinite sequence of truth assignments $\sigma = \langle \sigma_i, \sigma_{i+1}, \sigma_{i+2}, \ldots \rangle$ for the variables in $\mathcal{P}$, $\langle \sigma, i \rangle \models \varphi$ iff $\langle \sigma, i+1 \rangle \models \text{prog}(\sigma_i, \varphi)$.*

Proof sketch. Using induction and Theorem A.1, we can prove that the reward given by R_φ and r' is identical (at every time step) for any LTL formula $\varphi \in \Phi$, initial state $s_1 \in S$, and trace $s_1, a_1, ..., s_t, a_t$. Now, let's consider any state $s \in S$ and task $\varphi \in \Phi$. Given any optimal policy $\pi^*_\Phi(a|s, \varphi)$ for $\mathcal{M}_\Phi$, we can construct a policy $\pi_\varphi(a_t|s, a_1, ..., s_t, \varphi)$ for $\mathcal{M}_e$ that mimics the actions selection of $\pi^*_\Phi(a|s, \varphi)$ step by step. Hence, as the probability of reaching state s' given state s and action a is the same for $\mathcal{M}_\Phi$ and $\mathcal{M}_e$, both policies will induce the same probability distribution over traces and, as the reward functions are equivalent, both policies π^*_Φ and π_φ achieve the same expected discounted return. Finally, if we now have an optimal policy $\pi^*_\varphi(a_t|s, a_1, ..., s_t, \varphi)$ for $\mathcal{M}_e$, we can construct a non-stationary policy $\pi_\Phi(a_t|\langle s, \varphi \rangle, a_1, ..., \langle s_t, \varphi_t \rangle)$ for $\mathcal{M}_\Phi$ that mimics the actions selection of π^*_φ step by step. Following the same argument as before, we can see that π^*_φ and π_Φ achieve the same expected discounted return. Since $\mathcal{M}_\Phi$ is an MDP, we know that there exist a stationary policy $\pi'_\Phi(a|s, \varphi)$ that achieves at least as much return as any non-stationary policy $\pi_\Phi(a_t|\langle s, \varphi \rangle, a_1, ..., \langle s_t, \varphi_t \rangle)$. Therefore, we showed that optimal policies for $\mathcal{M}_\Phi$ are as good as optimal policies for $\mathcal{M}_e$ w.r.t. any R_φ (and vice versa). □

B. Experimental Details

In this section we provide some details on the task generation process as well as the hyperparameters used for our model training.

B.1. LTL Task Generation

Recall that we consider two task spaces: Partially-Ordered Tasks and Avoidance Tasks. The random generation of tasks is best described recursively with production rules of a context-free grammar.

Partially-Ordered Tasks

$$\begin{aligned} \textbf{formula} &\to \textbf{sequence} \wedge \textbf{formula} \mid \textbf{sequence} \\ \textbf{sequence} &\to \Diamond(\textbf{term} \wedge \textbf{sequence}) \mid \Diamond\textbf{term} \\ \textbf{term} &\to \mathsf{prop} \mid \mathsf{prop} \vee \mathsf{prop} \end{aligned}$$

In the above description, $\Diamond, \wedge, \vee$ are the *eventually*, *and*, *or* LTL operators, respectively and prop refers to any propositional variable.

Intuitively, Partially-Ordered Tasks presents k sequences of propositions which can be solved simultaneously. A trace is successful if and only if for every one of the k sequences, all the propositions in that sequence occur at some point in the trace (in the order of the sequence). Note that Partially-Ordered Tasks can never be falsified. However, most tasks are computationally intractable to solve in as few steps as possible due to the exponential number of possible solutions which must be considered. An example formula that is a conjunction of 2 sequences, each of depth 2 is:

$$\Diamond((A \vee B) \wedge \Diamond C) \wedge \Diamond(C \wedge \Diamond D))$$

In our Letter World experiments, the number of conjuncts was randomly sampled between 1 and 4, and the depth of each sequence was randomly sampled between 1 and 5. Each "term" had a 0.25 probability of being a disjunction of two propositions, and a 0.75 probability of being a single proposition.

To evaluate generalization to larger formulas, we considered (separately) increasing the depth of sequences and increasing the number of conjuncts. For increased depth tasks, the depth was 15 and the number of conjuncts was randomly sampled between 2 and 4. For increased number of conjuncts, the depth was randomly sampled between 3 and 5, and the number of conjuncts was 12.

Avoidance Tasks

$$\begin{aligned} \textbf{formula} &\to \textbf{sequence} \wedge \textbf{formula} \mid \textbf{sequence} \\ \textbf{sequence} &\to \neg\mathsf{prop}\ \mathsf{U}\ (\mathsf{prop} \wedge \textbf{sequence}) \mid \neg\mathsf{prop}\ \mathsf{U}\ \mathsf{prop} \end{aligned}$$

Table 2. PPO hyperparameters for `LetterWorld`. The same set of hyperparameters were used for both Avoidance and Partially-Ordered tasks.

	$\text{GNN}^{\text{pre}}_{\text{prog}}$	$\text{GRU}^{\text{pre}}_{\text{prog}}$	GNN_{prog}	GRU_{prog}	$\text{LSTM}_{\text{prog}}$	Myopic	GRU	No LTL
Env. steps per update		$\longleftarrow$		$2,048$		$\longrightarrow$		
Number of epochs	4	8	4	8	8	8	4	4
Minibatch Size	256	256	256	256	256	256	1,024	1,024
Discount factor (γ)		$\longleftarrow$		0.94		$\longrightarrow$		
Learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	10^{-4}	3×10^{-4}	3×10^{-4}
GAE-λ		$\longleftarrow$		0.95		$\longrightarrow$		
Entropy coefficient		$\longleftarrow$		0.01		$\longrightarrow$		
Value loss coefficient		$\longleftarrow$		0.5		$\longrightarrow$		
Gradient Clipping		$\longleftarrow$		0.5		$\longrightarrow$		
PPO Clipping (ε)		$\longleftarrow$		0.2		$\longrightarrow$		

Here, the $\neg, \mathsf{U}$ symbols are the *not*, *until* LTL operators, respectively. Similar to Partially-Ordered Tasks, several parallel sequences of propositions must be satisfied. However, this task space introduces the added challenge of propositions which must be avoided. The propositions to be avoided change as different parts of the task are solved. An example formula that is a conjunction of two sequences, each of depth two is:

$$(\neg A \,\mathsf{U}\, (K \wedge (\neg H \,\mathsf{U}\, J))) \wedge (\neg G \,\mathsf{U}\, (L \wedge (\neg F \,\mathsf{U}\, I)))$$

In order to guarantee that every formula can be solved, we do not allow the same proposition to appear twice in the same formula (avoiding conflicts such as $(\neg A \,\mathsf{U}\, A)$, which cannot be satisfied). In the Letter World, the number of conjuncts was randomly sampled between 1 and 2 and the depth of each sequence was randomly sampled between 1 and 3. For generalization to larger formulas, we considered depth 6 formulas with 1 conjunct (increased depth), as well as depth 2 formulas with 3 conjuncts (increased conjuncts). For the safety gym environment, we considered 1 conjunct, and randomly sampled the depth between 1 and 2 (longer tasks suffered from sparse reward, which is not the focus on this work).

B.2. Network Architectures

As mentioned in Section 4, we used PPO as the RL method for our experiments. We used the same actor (3 fully-connected layers with [64, 64, 64] units and ReLU activations) and critic (3 fully-connected layers with [64, 64, 1] units and Tanh activation) model for `LetterWorld` and `ZoneEnv`. In `LTLBootcamp` (pretraining), we used a single layer actor and critic with no hidden layers. This was to encourage the LTL module to learn a self-sufficient encoding (as the actor and critics cannot be transferred to downstream tasks). For discrete action space environments, the actor's output was passed through a logit layer before softmax. For the continuous case we assumed a Gaussian action distribution and parameterized its mean and standard deviation by sending the actor's output to two separate linear layers.

The Env Module is determined by the observation space of the underlying environment: in `LetterWorld` we used a 3-layer convolutional neural network with 16, 32 and 64 channels, kernel size of 2×2 and stride of 1 and in `ZoneEnv` we used a 2-layer fully-connected network with [128, 128] units and ReLU activations. Naturally, there is no Env Module for `LTLBootcamp`.

For LTL Module, we tested the following architectures with roughly the same number of parameters ($\sim 10^4$) to encode LTL formulas:

- *Graph Neural Networks (GNN):* The R-GCN architecture of Section 4 with $T = 8$ message passing steps and 32-dimensional node embeddings, i.e., $\boldsymbol{x}_v^{(t)} \in \mathbb{R}^{32}$. To reduce the number of trainable parameters we share the weight matrix across iterations for each edge type: $W_r = W_r^{(t)}$ $(0 \leq t \leq T)$. We observed better expressibility by concatenating the embedding of a node at iteration t with its one-hot encoding before passing it to the neighboring nodes for aggregation: $(\boldsymbol{x}_u^{(t)}, \boldsymbol{x}_u^{(0)})$, thus $W_r \in \mathbb{R}^{(32+32)\times 32}$. We used Tanh as the element-wise activation of Equation 2.
  Note that the embedding does not consider information from nodes more than T edges away from the root. However, this did not appear to be an issue in our experiments using $T = 8$, despite encountering formulas with ASTs larger than 8. One mitigating factor is that LTL progression generally reduces the size of formulas as parts of the task are solved, and the information most immediately relevant to the task tends to lie closer to the root.
- *Gated Recurrent Units (GRU):* A 2-layer bidirectional GRU with a 16-dimensional hidden layer.
- *Long Short-Term Memory (LSTM):* A 2-layer bidirectional LSTM with a 16-dimensional hidden layer.

We used `torch-ac`'s implementation of PPO (https://github.com/lcswillems/torch-ac).

Table 3. PPO hyperparameters for `ZoneEnv`. Only Avoidance tasks were considered on this environment.

	$\text{GNN}^{\text{pre}}_{\text{prog}}$	GNN_{prog}	Myopic
Env. steps per update	65,536	65,536	65,536
Number of epochs	10	10	10
Minibatch size	2,048	2,048	1,024
Discount factor (γ)	0.998	0.998	0.998
Learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}
GAE-λ	0.95	0.95	0.95
Entropy coefficient	0.003	0.003	0.003
Value loss coefficient	0.5	0.5	0.5
Gradient Clipping	0.5	0.5	0.5
PPO Clipping (ε)	0.2	0.2	0.2

B.3. PPO Hyperparameters

All experiments were conducted on a compute cluster using 1 GPU and 16 CPU cores per run. The hyperparameters used for PPO for each baseline are displayed in Table 2 for the `LetterWorld`, Table 3 for the `ZoneEnv`, and Table 4 for the `LTLBootcamp` (pretraining). Using a GNN architecture, training completed in approximately 26 hours in `LetterWorld` and 24 hours in the `ZoneEnv` (both for 20 million frames). Using a GRU to encode formulas was usually 2-3$\times$ more wall-clock efficient compared to GNN.

Note that all baselines which treat the problem as partially observable use an additional recurrent layer after the Env Model (i.e., *GRU* and *No LTL*). As backpropagation through all timesteps is computationally expensive, we backpropagate gradients only through the last 4 timesteps. We did not observe better performance by increasing the number of backpropagation steps.

C. Additional Results

C.1. Generalization to Unseen Objects

While the main focus of our work was generalization to new instructions, an important related problem is generalization to unseen *objects* (Hill et al., 2021; Leon et al., 2020). We conduct a simple experiment in `LetterWorld` to test object generalization in our framework by evaluating on unseen letters/propositions. While our framework normally uses one-hot encodings for propositions, such an approach is not conducive to generalization to new propositions. Here, we instead encode each letter as a random (but fixed), normalized vector in a low-dimensional space $\mathbb{R}^d$ (we use $d = 3$). Importantly, the same proposition is encoded in the same way in both the grid and in LTL formulas. We consider Avoidance tasks with a depth of 2 and train our agent on formulas over 12 fixed letters. We then evaluate this agent (over 5 seeds and 1000 episodes per seed) on formulas with the same structure, but over (a) 6 previously seen and 6 unseen letters, and (b) 12 unseen letters.

Table 4. PPO hyperparameters for `LTLBootcamp` (pretraining).

	GNN_{prog}	GRU_{prog}
(a) Avoidance Tasks		
Env. steps per update	8,192	8,192
Number of epochs	2	2
Minibatch size	1,024	1,024
Discount factor (γ)	0.9	0.9
Learning rate	10^{-3}	10^{-3}
GAE-λ	0.5	0.5
Entropy coefficient	0.01	0.01
Value loss coefficient	0.5	0.5
Gradient Clipping	0.5	0.5
PPO Clipping (ε)	0.1	0.1
(b) Partially-Ordered Tasks		
Env. steps per update	8,192	8,192
Number of epochs	2	4
Minibatch size	1,024	1,024
Discount factor (γ)	0.9	0.9
Learning rate	10^{-3}	3×10^{-3}
GAE-λ	0.5	0.95
Entropy coefficient	0.01	0.01
Value loss coefficient	0.5	0.5
Gradient Clipping	0.5	0.5
PPO Clipping (ε)	0.1	0.2

Table 5. RL agents are trained on LTL tasks over 12 letters, and are evaluated on tasks over some unseen letters. In each entry, we report the mean return over 5 seeds and 1000 episodes per seed, with 90% confidence error.

	% Unseen Letters	
	50%	100%
Ours	0.667 ± 0.015	0.607 ± 0.016
Random	-0.374 ± 0.021	-0.374 ± 0.021

Results are displayed in Table 5. Compared to a random action-selection baseline, our framework generalizes well to new tasks over unseen letters.

C.2. Pretraining Learning Curves

In Figure 6, we report the learning curves of GNN_{prog} and GRU_{prog} on the `LTLBootcamp` environment. Note that this is not meant to be an evaluation benchmark and is only used for pretraining the LTL module in our other experiments (see the main text, Figure 4). We observe, however, that the GNN is able to learn significantly faster than the GRU. Note that the Partially-Ordered tasks still remain extremely challenging to solve optimally, even in this abstracted environment.

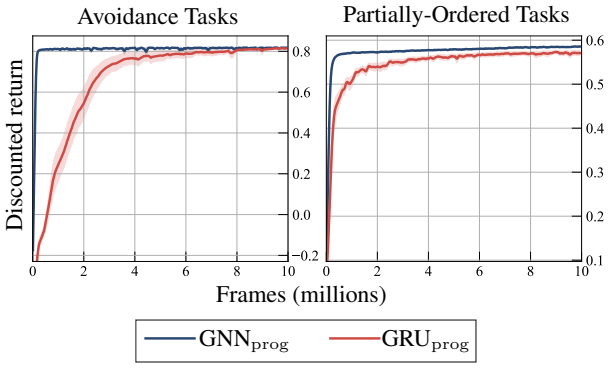

Figure 6. The learning curves of the GNN and GRU (both with progression) in the `LTLBootcamp` (pretraining) environment. Given random formulas, the task is to choose propositions which satisfy it in as few steps as possible. We report *discounted return* over the duration of training (averaged over 30 seeds, with 90% confidence intervals).