Sequence-to-Sequence Language Grounding of Non-Markovian Task Specifications

Nakul Gopalan*, Dilip Arumugam*, Lawson L.S. Wong, and Stefanie Tellex
Computer Science Department, Brown University, Providence, RI 02912
{ngopalan@cs., dilip_arumugam@, lsw@, stefie10@cs.}brown.edu
* The first two authors contributed equally

***Abstract*—Often times, natural language commands issued to robots not only specify a particular target configuration or goal state but also outline constraints on how the robot goes about its execution. That is, the path taken to achieving some goal state is given equal importance to the goal state itself. One example of this could be instructing a wheeled robot to "go to the living room but avoid the kitchen," in order to avoid scuffing the floor. This class of behaviors poses a serious obstacle to existing language understanding for robotics approaches that map to either action sequences or goal state representations. Due to the non-Markovian nature of the objective, approaches in the former category must map to potentially unbounded action sequences whereas approaches in the latter category would require folding the entirety of a robot's trajectory into a (traditionally Markovian) state representation, resulting in an intractable decision-making problem. To resolve this challenge, we use a recently introduced probabilistic variant of Linear Temporal Logic (LTL) as a goal specification language for a Markov Decision Process (MDP). While demonstrating that standard neural sequence-to-sequence learning models can successfully ground language to this semantic representation, we also provide analysis that highlights generalization to novel, unseen logical forms as an open problem for this class of model. We evaluate our system within two simulated robot domains as well as on a physical robot, demonstrating accurate language grounding alongside a significant expansion in the space of interpretable robot behaviors.**

I. Introduction

The broad spectrum of tasks that humans would like to see interpreted and executed by robots extends beyond realizing a particular goal configuration. A slightly richer space of tasks includes those that are looking to induce a constrained or repetitive robot execution. In the former category, a person may instruct a robot in the home to "go down the right side of the hallway and to the bedroom," restricting the space of possible paths the robot might take to reach its destination. Similarly, a command in the latter category may be "watch the sink for dirty dishes and clean any that you see," implying that the robot should enter a loop of repeatedly scanning for dirty dishes and cleaning any that appear. A robot system that can adequately represent and enable untrained users to express these complex behaviors would constitute a huge step forward in the area of human-robot interaction.

Existing approaches such as Linear Temporal Logic (LTL) [41, 35] allow these behaviors to be expressed as formal logical functions, enabling the automatic generation of robot controllers to execute these behaviors. While incredibly powerful, the average, non-expert user cannot be expected to hold the requisite knowledge for expressing arbitrarily complex behaviors via LTL. Consequently, we turn to natural language as a familiar interface through which non-expert users may convey their intent and desires while escaping the need for low-level programming knowledge. Unfortunately, due to the rich underlying semantics of LTL, there is no existing approach that takes open-ended natural language commands and maps directly to general LTL expressions.

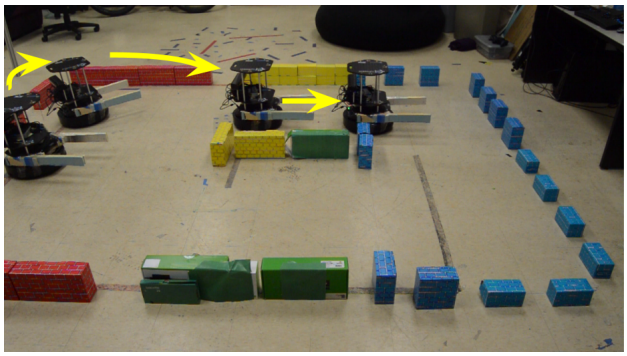


Fig. 1: In this robot demonstration we provided a natural language command of "Go through the yellow room to the blue room". The command is mapped to the LTL formula "$\Diamond(Y \wedge \Diamond B)$", which implies eventually satisfying the combination of going to yellow and subsequently reaching the blue room. In this work we map from natural language commands to such LTL formulae with sequence-to-sequence approaches.

This paper presents an approach for enabling a robot to learn a mapping between English commands and LTL expressions. We present the largest known dataset that maps between English and LTL in a model environment. We employ neural sequence-to-sequence learning models to infer the LTL sequence corresponding to a given natural language input sequence. While existing sequence-to-sequence models require large amounts of data to perform tasks at scale (such as English-French neural machine translation), we are not able to easily access such vast quantities of data due to the associated cost of obtaining annotations of natural language with their equivalent LTL logical forms. Consequently, we also outline a data collection pipeline through Amazon Mechanical Turk and employ data augmentation techniques to expand the size

of the parallel corpus. Finally, we conduct an analysis of our proposed sequence-to-sequence grounding models and their ability to generalize to novel natural language commands and generate LTL expressions never seen during training.

We evaluate our approach in simulation as well as using a real robot. We collected a corpus of more than 700 sentences mapping to LTL commands to a robot in a simulated 2D environment, which is the largest natural language to LTL corpus to the best of the authors' knowledge. We showed that we are able to correctly ground LTL expressions for a wide variety of commands. We demonstrate our approach on a simulated pick-and-place domain, and on a mobile robot (the Turtlebot). Our approach extends the space of achievable robot behaviors to include several non-Markovian objectives including repetitive patrolling, going through specified locations, and avoiding specified locations.

II. Related Work

The question of how to effectively convert between natural language instructions and robot behavior has been widely studied in previous work [50, 34, 24, 14, 9, 47, 8, 18, 11, 1, 33, 28, 36, 27, 40, 7, 2, 19, 37]. So far, there have been three categories of behavior specifications that these works have mapped natural language to: action sequences, goal states, and LTL specifications. They differ in terms of the following four desiderata:

- Environment-agnostic: The same natural language instruction may lead to different primitive actions being executed across different environments, or if the world is stochastic. Grounding language directly to action sequences is environment-dependent; the other two (goals and LTL) are not.
- Compact representation: Action sequences have length proportional to the trajectory, whereas goals and LTL formulae can be arbitrarily shorter.
- Non-Markovian specifications: Some typical robot behavior cannot be easily expressed as a goal state, such as "put down the knife before moving". This is trivial to specify in an action sequence, and is often expressible in LTL.
- Efficient planning: Action sequences require no planning and are therefore trivially efficient. For reaching goal states, efficient algorithms exist both within classical and decision-theoretic planning (MDPs) [38, 29]. Control policy synthesis for LTL is (doubly) exponential in the formula length in the worst case [12], although it is typically faster in practice, and many efficient fragments and approximations have been proposed.

Since we want to ground language for non-Markovian specifications in potentially novel and stochastic environments, we adopt LTL as our target specification in this paper, and will only review grounding language to temporal logic in the remainder of this section.

Kress-Gazit et al. [22, 24] pioneered the area of grounding language to LTL, although the initial work was limited to grounding a fragment termed "structured English". Subsequently, Dzifcak et al. [14] designed a combinatorial categorial grammar (CCG) that grounds a different fragment of English to the more-general computational tree logic (CTL*); however, this work suffers from the need to manually construct a grammar. Lignos et al. [28] instead used off-the-shelf parsers to interpret the full space of natural language, but do not provide a method to learn from new robot-specific language, and therefore their approach may be limited by the particular corpus used by the existing parser. In contrast, Boteanu et al. [7] collected a crowdsourced corpus of block-sorting instructions to train a Verifiable Distributed Correspondence Graph model that mapped natural language to structured English. In this work, we present a corpus that is an order of magnitude larger, and apply the sequence-to-sequence framework that allows grounding to the full space of LTL formulae (instead of the GR(1) fragment). In an orthogonal direction, Raman et al. [44] demonstrated the benefit of verifying LTL behavior by identifying unsatisfiable parts of LTL formulae and reporting which natural language commands resulted in these failures. Although verifiable behavior is a significant advantage of using LTL [23], we do not focus on verification in this work.

Modeling agent behavior with LTL has been of interest for a while [31, 24]. There have been various variants of LTL developed over the years for different reasons. We describe a few here, for a more detailed treatment please refer Kress-Gazit et al. [25]. To handle uncertainty, a probabilistic version of LTL, Probabilistic Computation Tree Logic [16] has been defined so as to evaluate over states of an MDP. Further, since LTL has an infinite time horizon, approaches have tried to shorten this time horizon so we verify properties to be valid for a certain time horizon [48]. Geometric LTL (GLTL) [30] is another such formulation for specifying goals for an MDP using LTL formuale, such that the formulae are only valid for a chosen time window. In this paper, we use GLTL to represent temporal behaviors because it allows planning with traditional MDP planners. However, our use of the neural sequence-to-sequence framework grants us flexibility, and we may also consider other variations of LTL to map natural language to, and perform non-Markovian behaviors. We leave exploration of other LTL variants for target specification from natural language to future work.

III. Approach

In order to fully specify our system for converting natural language to robot behavior, we begin by describing our problem setting and clarify its connection to the GLTL semantic representation inferred by our grounding model. We then go on to outline the recurrent neural network architectures used for mapping between natural language and GLTL expressions.

A. Problem Setting

We formalize the problem of language grounding within the context of an Object-oriented Markov Decision Process (OO-MDP). A Markov Decision Process (MDP) [5, 42] is a five-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma \rangle$ where $\mathcal{S}$ defines the robot state space, $\mathcal{A}$ specifies the actions available to the robot, $\mathcal{R}$ encodes the underlying task through numerical rewards defined for

each state-action pair, $\mathcal{T}$ defines the probabilistic transition dynamics of the environment, and γ is a discount factor. Given an MDP as input, a planning algorithm will produce a policy that maps from states to robot actions. Building on the MDP formalism, an OO-MDP [13] refines the notion of state to be a collection of object classes, each with their own set of attributes. Additionally, the OO-MDP framework includes a set of propositional logic functions, parameterized by instances of object classes, that can be used to query objects within the OO-MDP state.

Following the framework introduced by MacGlashan et al. [33], we treat natural language as the specification of a latent reward function that completes the definition of an otherwise fully-specified MDP. We use a language grounding model to arrive at a more consolidated, semantic representation of that reward function, thereby completing the MDP and allowing it to be passed to an arbitrary planning algorithm for generating robot behavior. More specifically, we think of each natural language command as specifying some latent LTL formula encoding the desired behavior. The OO-MDP propositional functions serve as possible atoms of the LTL expressions, creating an expressive language for defining tasks. In particular, LTL formulae are sufficiently expressive to subsume semantic representations used in previous goal-based language grounding work such as MacGlashan et al. [33] and Arumugam et al. [2].

B. Geometric linear temporal logic (GLTL)

Linear temporal logic has the following grammatical syntax:

$$\phi := \texttt{atom} \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \Box\phi \mid \Diamond\phi \mid \phi \mathcal{U} \psi \mid \bigcirc \phi$$

$\texttt{atom}$ denotes an atomic proposition; $\neg$, $\wedge$, and $\vee$ are logical "not", "and", and "or"; $\Box$ denotes "always", $\Diamond$ denotes "eventually", $\mathcal{U}$ denotes "until", and $\bigcirc$ denotes "next". Semantic interpretations of temporal logic operators can be found in Manna and Pnueli [35].[1]

Following work done by Bacchus et al. [3] for specifying reward functions over temporal sequences via LTL, Littman et al. [30] introduced the geometric LTL (GLTL) variant that replaces the standard LTL operators with "soft" substitutes. Effectively, these probabilistic operators are equivalent to their hard LTL counterparts but satisfying each operator is restricted to some bounded window of time as determined by a draw from a geometric distribution. More concretely, instead of $\Box p$, representing that p always holds true indefinitely, GLTL would have $\Box_\mu p$ for indicating that p holds for the next $k \sim \text{Geom}(\mu)$ timesteps. Similarly, $\Diamond_\mu p$ and $q\mathcal{U}_\mu p$ correspond to p becoming true in the next k timesteps and q holding true at least until p becomes true in the next k timesteps respectively where, again, $k \sim \text{Geom}(\mu)$. While sacrificing the guarantees and proofs of correctness typical of LTL, the probabilistic nature of GLTL enables learning while specifying rewards in a generalizable, environment-independent fashion. We include more on how GLTL ties into our approach in Section III-A and, for more information on GLTL itself, please consult [30].

Crucially, Littman et al. [30] connect GLTL back to MDPs in a way that not only specifies the desired behavior without overfitting to a single environment instance by folding the stochastic semantics of the GLTL expression into the stochastic transitions of the environment, enabling the application of standard reinforcement learning and planning techniques. In particular, each atomic proposition of a GLTL expression has an associated three-state MDP consisting of an initial, accepting, and rejecting state. From the initial state, there is a single action in this *specification MDP* that will transition to the accepting state if the proposition holds and move to the rejecting state otherwise. Each GLTL operator represents some fixed transformation or combination of these MDPs resulting in a new specification MDP. Once an inferred GLTL expression is converted to its corresponding specification MDP, it is combined with the separate environment MDP resulting in an MDP whose solution represents a policy capturing the desired behavior. For the full details of specification MDP construction and combination with an environment MDP, please see [30]. We model the procedure of mapping from natural language to GLTL expressions as a neural machine translation problem.

Although we choose GLTL paired with an MDP to find policies corresponding to LTL expressions, our language grounding system can work with any framework for computing a satisfying controller for the robot, such as those described in the related work [25]. Notice that the switch is commensurate with defining a new machine translation problem with a target language defined by the syntax of the alternative framework.

C. Mapping Language to GLTL

In order to handle the task of translating from natural language to GLTL expressions, we turn to recent neural-network architectures for sequence learning that have already proven to be incredibly performant in other machine translation tasks. We are presented with a sequence taken from some source language $\mathbf{x} = [x_1, \ldots, x_N]$ and would like to infer a corresponding sequence of some target language $\mathbf{y} = [y_1, \ldots, y_M]$. Given a translation model with parameters θ, we look to identify the most likely target sequence and decompose its corresponding probability into the product of partial translation probabilities:

$$p(\mathbf{y}|\mathbf{x}, \theta) = \prod_{m=1}^{M} p(y_m|\mathbf{x}, \mathbf{y}_{<m}, \theta) \tag{1}$$

where $\mathbf{y}_{<m} = [y_1, \ldots, y_{m-1}]$.

Treating the space of natural language commands as a source language and GLTL expressions as a target language, we collect a parallel corpus and optimize the neural-network architecture parameters θ (see Section IV-C for more details on the data collection procedure). Specifically, we leverage the recurrent neural network (RNN) encoder-decoder framework [10, 46] extended by Bahdanau et al. [4].

[1] In this work we do not collect data on behavior requiring the "next" operator, and therefore it never appears in our grounded formulae; however, the same framework can be used if relevant data for "next" is collected.

Widely used across a variety of natural language tasks, RNNs are models that map sequential inputs to high-dimensional output spaces, using recurrent cells that maintain an internal or hidden-state representation of the sequence processed thus far [17, 10, 15]. Neural sequence-to-sequence learning use two distinct RNN models, an encoder and decoder, to map between source and target language sequences. An encoder processes an input sequence and, for each input token, produces an output and an updated hidden state. While the hidden state stores a global summary of the overall sequence, the output encodes local information computed from the current hidden state and the current input token. After all input tokens are fed to the encoder, there is a resulting sequence of encoder outputs and hidden states. Subsequently, a decoder model generates target language sequences by mapping an input target language symbol to a probability distribution over the target language vocabulary [6]. Connecting the two models is the final hidden state of the encoder RNN cell (that is, the hidden state after processing the entire input sequence) which is used to initialize the hidden state of the decoder RNN cell [10, 46]. The encoder hidden state is initialized with an all zeros vector and a special start token is fed as the first input to the decoder in order to initialize decoding.

In both the encoder and decoder of our model, we use the Gated Recurrent Unit (GRU) [10] as the core RNN cell. Briefly, a GRU is one type of RNN cell that, using the previous hidden state s_{t-1} and current input x_t, performs the following computations:

$$\text{Reset gate:} \quad r_t = \sigma(\mathbf{W_r} \cdot x_t + \mathbf{U_r} \cdot s_{t-1} + \mathbf{b_r}) \tag{2}$$

$$\text{Output:} \quad g_t = \tanh(\mathbf{W_g} \cdot x_t + \mathbf{U_g} \cdot (r_t \odot s_{t-1}) + \mathbf{b_g}) \tag{3}$$

$$\text{Update gate:} \quad z_t = \sigma(\mathbf{W_z} \cdot x_t + \mathbf{U_z} \cdot s_{t-1} + \mathbf{b_z}) \tag{4}$$

$$\text{State update:} \quad s_t = (1 - z_t) \odot s_{t-1} + z_t \odot g_t \tag{5}$$

resulting in an output vector g_t and a new hidden state s_t where $(\cdot)$ and $(\odot)$ denote matrix-vector and Hadamard products respectively. Intuitively, r_t is a "reset" gate affecting how hidden state information is combined with the current input to produce the cell output, g_t, and z_t is an "update" gate negotiating how much information is preserved within the hidden state. All parameters in bold denote trainable parameters of the cell that are optimized during training of the entire architecture via backpropagation.

Given the GRU encoder and decoder, f_{enc} and f_{dec}, words of the input sequence are first mapped to their corresponding vector representations via an embedding lookup. Intuitively, since the individual tokens can simply be represented as integers, the word embeddings provide a high-dimensional representation that is optimized as part of the neural network and can be used to capture semantic information about each individual word. Feeding each input embedding, $\hat{\mathbf{x}}_\mathbf{j}$, in sequence produces a corresponding sequence of encoder outputs, $\left[h_1, \ldots, h_N\right]$, and hidden states $\left[s_1, \ldots, s_N\right]$. where each h_j, s_j comes from:

$$h_j, s_j = f_{enc}(s_{j-1}, \hat{\mathbf{x}}_j) \tag{6}$$

Once the input sequence has been processed, the final encoder hidden state, s_N is used to initialize the RNN cell of the decoder. During decoding, the previously inferred token of the output sequence is mapped to a probability distribution over the target vocabulary from which the next output token is sampled:

$$p(y_i|\mathbf{x}, \mathbf{y}_{<m}, \theta) = f_{dec}(v_{i-1}, \hat{\mathbf{y}}_{i-1}) \tag{7}$$

Here $\hat{\mathbf{y}}_{i-1}$ denotes the embedding for the previously decoded token, $\hat{y}_{i-1}$. In all of our experiments, we use a greedy decoding scheme and treat the token in the distribution with highest probability as the inferred token, $\hat{y}_i$.

Subsequent work by Bahdanau et al. [4] proposed learning a weighting or attention scheme to selectively utilize encoder output information during decoding. The resulting Bahdanau attention mechanism reparameterizes the decoder as a function of the previous token embedding and a context vector. At each step of decoding, the context vector weights the information of the encoder outputs $\left[h_1, \ldots, h_N\right]$ according to:

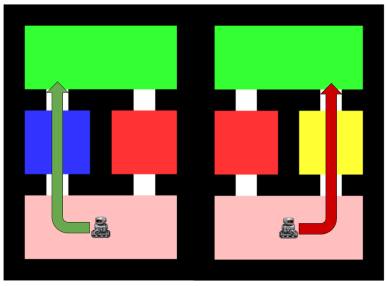
$$c_i = \sum_{j=1}^{N} \alpha_{ij} h_j \tag{8}$$

where α_i is a weight on the information carried in h_j. The individual attention weights are computed as outputs of a feedforward neural network, $a(v_i, \left[h_1, \ldots, h_N\right])$, with a final softmax activation and where v_i is the current hidden state of the decoder RNN cell. Accordingly, decoding occurs by greedily sampling the next token distribution:

$$p(y_i|\mathbf{x}, \mathbf{y}_{<m}, \theta) = f_{dec}(v_{i-1}, \hat{\mathbf{y}}_{i-1}, c_i) \tag{9}$$

Even with alternatives to Bahdanau attention, attention weights are typically computed as a function of the current decoder hidden state and the encoder outputs [32]. Given the rigid structure of GLTL as a target output language, we examine an alternative attention mechanism that computes attention weights purely as a function of decoder parameters. More formally, we propose an *encoder-agnostic* attention mechanism where each attention weight α_i is computed by a feedforward neural network $a(\hat{\mathbf{y}}_{i-1}, v_i)$ that takes the previously decoded token embedding and the current decoder hidden state as inputs. This attention scheme captures the idea that the previously decoded token and current state of the target translation generated thus far offer a clearer signal for weighting the encoder outputs than the encoder outputs themselves. Although seemingly counterintuitive, we consider a particular scenario where the sequence-to-sequence grounding model must generalize to a language command at test time that corresponds to a novel GLTL expression never seen during training. Instead of being subject to the learned mechanics of the source language that may vary dramatically at test time, the encoder-agnostic attention scheme would maintain focus on the GLTL target language side that exhibits significantly less variation and follows a small, well-defined lexicon.

Unlike the architecture outlined in Bahdanau et al. [4], we found a single, forward RNN encoder was sufficient for our



(a) Static mobile-manipulation domain images presented to AMT users for annotation. Positive (green) and negative (red) labels were applied to the images so that AMT users could infer the constraint reflected in the robot's behavior.

Example Command	GLTL Expression
Go to the green room.	$\Diamond G$
Go into the red room.	$\Diamond R$
Enter blue room via green room.	$\Diamond(G \wedge \Diamond B)$
Go through the yellow or red room, and enter the blue room	$\Diamond((R \vee Y) \wedge \Diamond B)$
Go to the blue room but avoid the red room.	$\Diamond B \wedge \neg\Box R$
While avoiding yellow navigate to green.	$\Diamond G \wedge \neg\Box Y$
Scan for blocks and insert any found into bin.	$\Box((S\mathcal{U}\neg A) \wedge \Diamond A)$
Look for and pick up any non red cubes and put them in crate.	$\Box((S\mathcal{U}\neg N_R) \wedge \Diamond N_R)$

(b) Example commands and corresponding GLTL formulas. R, G, and B are propositions in Cleanup World for testing if the agent is in the Red, Green or Blue rooms respectively. S, A and N_R are propositions for the pick-and-place domain. S tests if the table has been scanned once; A tests if all blocks are in the bin; and N_R test if all blocks expect the red colored ones are in the bin.

Fig. 2: Image from Cleanup World shown for data collection, and sample commands collected using Amazon Mechanical Turk (AMT) for the domains of Cleanup World and Pick and place.

task and opted not to use a bi-directional RNN encoder in favor of reduced training time and sample complexity. All models were implemented in PyTorch [39] and trained using the Adam optimizer [20] with a learning rate of 0.001. Embedding and RNN output sizes were set to 50 and 256 units respectively. Additionally, we made use of dropout regularization [45] before and after both the encoder and decoder GRUs with a keep probability of 0.8. We found that reversing all input sequences provided a small increase in grounding accuracy. For each training sample, a random choice was made between providing the ground truth output token, y, (teacher forcing [49]) and providing the actual decoded symbol, $\hat{y}$, with 0.5 probability.

IV. Experiments

We now outline the two domains used for evaluating our approach as well as the details of our data collection procedure for training the three presented grounding models.

A. Mobile-Manipulation Robot Domain

Cleanup World [33] consists of a single robot agent acting within an environment characterized by distinctly colored rooms and movable objects. We chose this domain for our experiments as it allows us to express a wide variety of constrained execution tasks. For the purposes of our experiments, these restrictions took on one of two forms: the robot would need to reach it's target destination by either passing through or explicitly avoiding a particular room in the environment.

B. Pick-and-place domain

In order to showcase instances of repetitive robot execution, we designed a pick-and-place domain where a robot is meant to patrol the environment waiting for sudden events that trigger some desired behavior. More concretely, the domain is defined by distinct table and bin regions where colored blocks may reside. These blocks have only two attributes: first for location, table or bin; second for color where the blocks can be red or green or blue or yellow. Initially, there may not be any blocks present and so the robot must utilize a scan action to survey the area until an appropriately colored block can be picked up and moved from the table to the bin. A scan operation reveals a new block, if it was "placed" on the table. The placement can be assumed to be done by a human user. The pick-and-place action picks a block and moves it to the bin. Note that the reachable state space of this domain grows rapidly as the number of blocks on the table increases, as each block can be at different locations. Given an agent with only two scan and pick-and-place actions at its disposal, we experimented with two core task types: placing all blocks found during scanning into the bin or placing only blocks of a certain chosen color into the bin.

C. Data-collection Procedure

An Amazon Mechanical Turk (AMT) user study was conducted in order to collect data for training our neural sequence-to-sequence grounding models mapping natural language commands to GLTL expressions in each of our domains. To construct the parallel corpus, annotators were presented with static images (for Cleanup World) and short video clips (for pick-and-place) depicting a particular robot behavior as shown in https://streamable.com/8lqab. Users were then asked to provide a single sentence instruction that would induce the observed behavior. For the mobile-manipulation robot domain, sample images provided to AMT annotators can be seen in Figure 2a. Specifically, these images were displayed to users with positive (green arrow) and negative (red arrow) labels so that annotators could infer the constraint being placed on the robot's execution.

Curiously, we found that annotators were more inclined to specify positive over negative behavior in their instructions. For instance, an attempt to collect data for behavior matching the command "go to the green and avoid the yellow room"

	Domain #1		Domain #2	
	Original	Expanded	Original	Expanded
Seq2Seq	$93.10 \pm 1.60\%$	$95.25 \pm 0.30\%$	$86.09 \pm 1.03\%$	$93.42 \pm 0.96\%$
Seq2Seq + Bahdanau Attention	$\mathbf{93.45 \pm 0.60}\%$	$\mathbf{95.51 \pm 0.11}\%$	$\mathbf{87.15 \pm 0.36}\%$	$93.78 \pm 0.29\%$
Seq2Seq + Encoder-Agnostic Attention	$93.18 \pm 0.94\%$	$94.98 \pm 0.30\%$	$86.47 \pm 0.72\%$	$\mathbf{93.92 \pm 0.44}\%$

Fig. 3: Accuracy and standard deviation of 5-fold cross validation for each grounding model and domain, averaged over 3 independent runs.

would often result in commands where users would instruct the robot to navigate to the green room utilizing whichever rooms were designated under the positively labeled images. Although technically correct, these instructions pose an obstacle as the intended, ground-truth GLTL expression (containing, for example, the token for the yellow room) would never include a symbol with semantic meaning associated with the rooms mentioned (for example, the blue or red rooms). In order to address this problem, a manual relabeling of data was performed such that samples whose instruction did not align to the intended GLTL formula were instead mapped to the corresponding GLTL formula consistent with the instruction. Filtering commands that reflected a clear misunderstanding of the annotation task resulted in parallel corpora consisting of 501 and 345 commands for the mobile-manipulation and patrol domains respectively. In aggregate, these commands reflected a total of 15 and 2 unique GLTL expressions respectively. Examples of natural language commands and their corresponding GLTL formulae can be seen in Figure 2b.

In order to supply additional data for the mobile-manipulation domain, we utilized a subset of the Cleanup World dataset collected by Arumugam et al. [2] consisting of 356 agent navigation and block manipulation commands, swapping their Markov reward function representation for the GLTL equivalent. Together, the combined dataset denotes the *original dataset* for the mobile manipulation domain used throughout all of our experiments. The pick-and-place domain original dataset received no extra commands. To further expand on the data present for learning across both domains, a synthetic data expansion procedure was applied to both datasets. Excluding the minority of commands pertaining to block manipulation behavior, all other commands were mapped to new commands through the substitution of color words in the natural language and the equivalent swapping of atoms in the corresponding GLTL expressions. For example, the command "move to the red room" and corresponding expression $\Diamond R$ would be mapped to three new language commands (one for each of the blue, green and yellow rooms) along with the corresponding GLTL expressions ($\Diamond B$, $\Diamond G$, $\Diamond Y$). This expansion resulted in two *expanded datasets* for each of the domains consisting of 3382 and 745 commands respectively reflecting a total of 39 and 5 unique GLTL expressions.

D. Language Grounding

We conducted 5-fold cross validation experiments across three grounding models and present the language grounding accuracy means and standard deviations in Table 3. Results were averaged over 3 independent trials with distinct random seeds. For each instance, correctness was determined by comparing the complete, greedily-decoded GLTL formula to ground truth. Training was done using two independent parallel corpora, one for each of the domains outlined in above. Additionally, we report results on both the original datasets, consisting of natural language commands exactly as collected through AMT, and the expanded datasets synthetically generated via the procedure outlined in Section IV-C. Notably, we find that all three models exhibit roughly identical performance despite the use of two different attention mechanisms. We suspect that the lack of an effect is due to the dramatically smaller vocabulary size of GLTL by comparison to traditional neural machine translation problems. However, we do find the use of an attention mechanism to have some effect on enabling generalization and the inference logical forms not seen during training.

In order to establish how well each grounding model captures the semantics of natural language and GLTL expressions, we conducted an experiment to assess the capacity for each model to infer novel GLTL expressions. Focusing on the mobile manipulation domain, we randomly sampled varying percentages of the 39 unique GLTL expressions represented across the collected within the expanded corpus of 3382 commands. All samples in the parallel corpus associated with the random sampled commands were used as training data while the entire remainder of the corpus was treated as a held-out test set consisting only of GLTL expressions not seen during training. The results of the experiment are shown in Figure 4 with error bars denoting 95% confidence intervals computed over 10 independent trials. On average, we find that our encoder-agnostic attention scheme is slightly more adept at generalizing to novel commands. The results altogether, however, suggest that this type of generalization is still an open challenge for these models that traditionally excel in standard neural machine translation tasks that enjoy access to vast quantities of parallel training data. We believe that adapting these techniques to better operate in the extremely low resource area of robot language learning is an important direction for future research.

V. Results

A. Language Grounding

Our results confirm that a standard neural sequence learning model, without the use of an attention mechanism, is sufficient for achieving highly accurate language grounding to GLTL logical forms. We do, however, witness the benefits of attention when faced with the challenge of generalizing beyond the space of GLTL expressions seen at training time. Although, on average, our encoder-agnostic attention scheme does represent a fairly substantial improvement, it still leaves much to be desired. This problem of generalization without any training experience (or zero-shot generalization) within neural sequence-to-sequence models is further studied by Lake and Baroni [26] who conduct a series of experiments assessing various dimensions on generalization within a new domain for mapping navigation commands to action sequences. Notably, their study finds that generalization is only reliable when novel test time constituents are observed within a variety of contexts during training. As these neural-based translation approaches continue to improve, the question of how to make them more amenable to sparse-data, robot-learning settings will become increasingly important.

Beyond the challenge of generalization, there were a few other sources of error across the three grounding models. In the mobile-manipulation domain corpus, the longest language command consisted of 27 tokens. Demonstrating another well-known challenge of sequence learning and RNN models in general, these longer, outlier sequences posed a challenge and often produced invalid GLTL expressions. Despite the use of input sequence reversal to help combat this challenge, increasing sequence length forces the RNN cell to carry information across a larger number of timesteps; after a point, the degradation in the latent semantic information makes accurate translation incredibly difficult. Additionally, we found instances where the models produced a correct GLTL formula type but had the atoms reversed (such as selecting to avoid the target destination room and navigate towards the avoidance room). These commands tended to have minor grammatical errors or lacked certain keywords to indicate the blue *room* or the yellow *area*.

More generally, we note that the previously discussed issues with our approach are only a subset of a much broader space of challenges inherited from neural machine translation. The full space of challenges is perhaps most succinctly and carefully explored in the work of Koehn and Knowles [21] who outline six key deficiencies of general neural machine translation. While these challenges are made quite apparent from the scale of typical translation problems (mapping between millions of vocabulary tokens) a few issues of particular importance to the robotics community include out-of-domain words, low-resource translation, low-frequency words, and long sentences. Note that, in this work, we have already demonstrated and discussed the difficulties of low-resource translation (that is, translation with limited training data) alongside low-frequency words. Given the high-cost of acquiring annotations, robotics datasets are often built within a particular context and geared towards a specific domain. As the demand for these systems grows and requires a single system to operate across domains (for instance, the home and the workplace), the inability for current translation models to handle identical words with context-sensitive translations (or out-of-domain words) will prove to be a bottleneck. Moreover, as commands naturally grow in complexity and encode an increasing number of tasks, so too will the corresponding output sequences, resulting in decreased translation/grounding accuracy. Given the clear mutual interest, a rich direction of future work includes collaborating with the natural language processing community to develop solutions to these problems and bring them to bear on robotics domains.

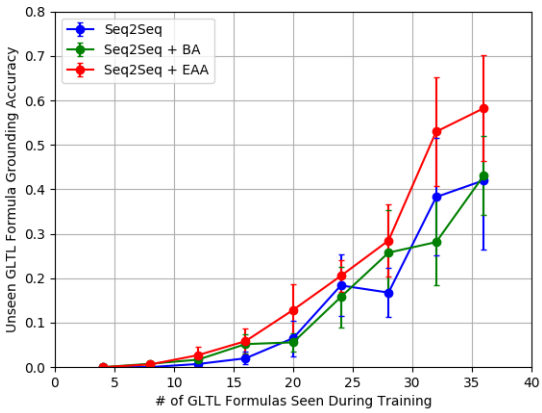


Fig. 4: Grounding accuracies of various sequence-to-sequence models evaluated on held-out subsets of the training data consisting entirely of novel GLTL expressions. Error bars represent 95% confidence intervals computed over 10 independent runs.

B. Pick-and-place simulation

We implemented the pick-and-place domain in simulation and observed the behavior of the two command types. The first command type was to move any block that is placed on the table to the bin. This behavior was highly consistent across repeated trials. The agent would scan, and a block might be placed during this scan operation. The agent would detect the placed block during the scan operation, and would choose to place the block in one of the following time steps. Conversely, we found that the behavior for selectively placing blocks of a specific color in the bin experienced difficulties during the execution of the inferred GLTL formula.

As described in Section II a GLTL expression holds true only for certain time steps, as planning within MDPs for an infinite time horizon compromises learnability and performing value backups for planning. Specifically for this task the agent placed non-red blocks into the bin and continued looking for more blocks. We noticed that the agent initially (and correctly) does not pick any red blocks it finds; however, as the number

of blocks revealed to the robot increases, it becomes more likely that the robot picks up a red block, as the formula and its corresponding behavior would only be true for a certain number of time steps dependent on the geometric discounting. With a discount factor of 0.99 we noticed that the agent would pick up a red block after about $5-10$ blocks were revealed to the agent when scanning. However, the agent constantly placed more non-red blocks in the bin, even when the number of red blocks was more than the number of rest of the blocks. An easy thing to increase the duration of placing non-red blocks would be to increase the discount factor, however this would lead to increased planning time, as the agent is planning over a longer horizon.

This simulation domain shows that the GLTL formulation can easily handle behaviors that include repetitive subtasks while being able to react to certain, pre-conditioned events. The experiment also helps us understand some of the limitations of GLTL, as an approximation of LTL for a chosen time horizon. Future work includes implementing our simulated pick-and-place domain on a real robot.

C. Robot Demonstration

To further demonstrate the efficacy of our approach, we translated our mobile-manipulation domain into the physical world with a Turtlebot agent. The problem space focused on constrained execution tasks requiring the robot the enter a goal room while either avoiding or passing through specific rooms. As in simulation, our physical setup consisted of four rooms uniquely identified by color and the agent's position in the world was tracked by a motion capture system. Using the Robot Operating System (ROS) [43] speech to text API, we converted speech utterances to natural language text that could be passed to a trained instance of our grounding model with Bahdanau attention, producing a GLTL formula. Treating the Cleanup World MDP as the environment MDP and identifying the specification MDP of the GLTL formula, we combine the two MDPs and apply a standard MDP planning algorithm to generate robot behavior in real time. The planning is real time as the formulae for these tasks are not very long and GLTL allows fast planning, most of the delays observed in the video are networking delays when using speech-to-text. The primitive north, south, east, west actions of the Cleanup Domain agent were converted into a series of low level ROS Twist messages using hand-written controllers. A video of our robot accurately responding to user commands in real time is provided as supplementary material and uploaded at https://streamable.com/f7jet.

VI. Conclusion

This paper demonstrates an approach for mapping between English commands and LTL expressions through neural sequence-to-sequence learning models. We presented techniques for data augmentation and a novel attention mechanism that enables the system to map between novel English commands and novel LTL expressions not encountered at training time so long as the constituent LTL atoms have been previously observed. We demonstrated this approach within two domains for mobile-manipulation and pick-and-place tasks as well as on a physical robot.

Directions for future work include the exploration of methods for learning compositional models of semantics that better utilize the underlying parse structure of the target language. In exploiting this structure the hope is to achieve greater success at generalization without incurring an additional sample complexity cost where data is so costly to acquire. An alternative perspective is to more directly tackle the zero-shot generalization problem of neural sequence-to-sequence methods, perhaps through further exploitation of attention mechanisms or some alternate network architecture. Furthermore, although this work primarily focuses on the task inference side of language grounding, opportunities certainly exist to consider the conjunction between these sequence learning models and task execution components. Simple steps in this direction could include integrating hierarchical planning frameworks as in Arumugam et al. [2] or leveraging an alternate system for identifying the best choice of planner based on the inferred specification type.

Acknowledgments

This material is based upon work supported by the National Science Foundation under grant number IIS-1637614, DARPA under grant numbers W911NF-10-2-0016 and D15AP00102, and NASA grant number NNX16AR61G.

Lawson L.S. Wong was supported by a Croucher Foundation Fellowship.

References

[1] Jacob Andreas and Dan Klein. Alignment-based compositional semantics for instruction following. In *Empirical Methods in Natural Language Processing*, 2015.

[2] Dilip Arumugam, Siddharth Karamcheti, Nakul Gopalan, Lawson L.S. Wong, and Stefanie Tellex. Accurately and efficiently interpreting human-robot instructions of varying granularities. *Robotics: Science and Systems*, 2017.

[3] Faheim Bacchus, Craig Boutilier, and Adam Grove. Rewarding behaviors. In *National Conference on Artificial Intelligence*, pages 1160–1167, 1996.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2014.

[5] R. Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, 6:679–684, 1957.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2000.

[7] A. Boteanu, T. M. Howard, J. Arkin, and H. Kress-Gazit. A model for verifiable grounding and execution of complex natural language instructions. In *IEEE/RSJ*

International Conference on Intelligent Robots and Systems, 2016.

[8] Daniel J. Brooks, Constantine Lignos, Cameron Finucane, Mikhail S. Medvedev, Ian Perera, Vasumathi Raman, Hadas Kress-Gazit, Mitch Marcus, and Holly A. Yanco. Make it so: Continuous, flexible natural language interaction with an autonomous robot. In *AAAI Conference on Artificial Intelligence Workshop on Grounding Language for Physical Systems*, 2012.

[9] David L. Chen and Raymond J. Mooney. Learning to interpret natural language navigation instructions from observations. In *AAAI Conference on Artificial Intelligence*, 2011.

[10] Kyunghyun Cho, Bart van Merrienboer, aglar Gülehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Empirical Methods in Natural Language Processing*, 2014.

[11] I. Chung, O. Propp, M. R. Walter, and T. M. Howard. On the performance of hierarchical distributed correspondence graphs for efficient symbol grounding of robot instructions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[12] Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the Association for Computing Machinery*, 42(4):857–907, 1995.

[13] Carlos Diuk, Andre Cohen, and Michael L. Littman. An object-oriented representation for efficient reinforcement learning. In *International Conference on Machine Learning*, 2008.

[14] Juraj Dzifcak, Matthias Scheutz, Chitta Baral, and Paul W. Schermerhorn. 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 *IEEE International Conference on Robotics and Automation*, 2009.

[15] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[16] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 98:1735–80, 1997.

[18] T. M. Howard, S. Tellex, and N. Roy. A natural language planner interface for mobile manipulators. In *IEEE International Conference on Robotics and Automation*, 2014.

[19] Siddharth Karamcheti, Edward C. Williams, Dilip Arumugam, Mina Rhee, Nakul Gopalan, Lawson L.S. Wong, and Stefanie Tellex. A tale of two DRAGGNs: A hybrid approach for interpreting action-oriented and goal-oriented instructions. *Annual Meeting of the Association for Computational Linguistics Workshop on Language Grounding for Robotics*, 2017.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[21] Philipp Koehn and Rebecca Knowles. Six challenges for neural machine translation. In *NMT@ACL*, 2017.

[22] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. From structured English to robot motion. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.

[23] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[24] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Translating structured english to robot controllers. *Advanced Robotics*, 22(12):1343–1359, 2008.

[25] Hadas Kress-Gazit, Morteza Lahijanian, and Vasumathi Raman. Synthesis for robots: Guarantees and feedback for robot behavior. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1), 2018.

[26] Brenden M. Lake and Marco G Baroni. Still not systematic after all these years: On the compositional skills of sequence-to-sequence recurrent networks. *CoRR*, abs/1711.00350, 2017.

[27] Percy Liang. Learning executable semantic parsers for natural language understanding. *Communications of the Association for Computing Machinery*, 59(9):68–76, 2016.

[28] C. Lignos, V. Raman, C. Finucane, M. Marcus, and H. Kress-Gazit. Provably correct reactive control from natural language. *Autonomous Robots*, 38(39), 2015.

[29] Michael L. Littman, Thomas L. Dean, and Leslie Pack Kaelbling. On the complexity of solving Markov decision problems. In *Uncertainty in Artificial Intelligence*, 1995.

[30] Michael L. Littman, Ufuk Topcu, Jie Fu, Charles Lee Isbell, Min Wen, and James MacGlashan. Environment-independent task specifications via GLTL. *CoRR*, abs/1704.04341, 2017.

[31] S. Loizou and K. Kyriakopoulos. Automatic synthesis of multi-agent motion tasks based on ltl specifications. *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, 1:153–158 Vol.1, 2004.

[32] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Empirical Methods in Natural Language Processing*, 2015.

[33] James MacGlashan, Monica Babes-Vroman, Marie desJardins, Michael L. Littman, Smaranda Muresan, Shawn Squire, Stefanie Tellex, Dilip Arumugam, and Lei Yang. Grounding english commands to reward functions. In *Robotics: Science and Systems*, 2015.

[34] Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. In *National Conference on Artificial Intelligence*, 2006.

[35] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems - specification*. Springer-

Verlag New York, 1992.

[36] Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *AAAI Conference on Artificial Intelligence*, 2016.

[37] Dipendra Kumar Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning. In *Empirical Methods in Natural Language Processing*, 2017.

[38] Christos Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[39] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *Neural Information Processing Systems Workshop on The Future of Gradient-based Machine Learning Software & Techniques*, 2017.

[40] Rohan Paul, Jacob Arkin, Nicholas Roy, and Thomas M. Howard. Efficient grounding of abstract spatial concepts for natural language interaction with robot manipulators. In *Robotics: Science and Systems*, 2016.

[41] Amir Pnueli. The temporal logic of programs. In *IEEE Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[42] Martin L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.

[43] Morgan Quigley, Josh Faust, Tully Foote, and Jeremy Leibs. ROS: an open-source robot operating system. In *IEEE International Conference on Robotics and Automation Workshop on Open Source Software*, 2009.

[44] Vasumathi Raman, Constantine Lignos, Cameron Finucane, Kenton Lee, Mitch Marcus, and Hadas Kress-Gazit. Sorry dave, i'm afraid i can't do that: Explaining unachievable robot tasks using natural language. In *Robotics: Science and Systems*, 2013.

[45] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[46] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Neural Information Processing Systems*, 2014.

[47] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI Conference on Artificial Intelligence*, 2011.

[48] Cristian-Ioan Vasile, Derya Aksaray, and Calin Belta. Time window temporal logic. *Theoretical Computer Science*, 691:27–54, 2017.

[49] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.

[50] Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Uncertainty in Artificial Intelligence*, 2005.