Robotics: Science and Systems 2020
Corvalis, Oregon, USA, July 12-16, 2020

Grounding Language to Non-Markovian Tasks with No Supervision of Task Specifications

Roma Patel, Ellie Pavlick and Stefanie Tellex

***Abstract*—Natural language instructions often exhibit sequential constraints rather than being simply goal-oriented, for example "go around the lake and then travel north until the intersection". Existing approaches map these kinds of natural language expressions to Linear Temporal Logic expressions but require an expensive dataset of LTL expressions paired with English sentences. We introduce an approach that can learn to map from English to LTL expressions given only pairs of English sentences and trajectories, enabling a robot to understand commands with sequential constraints. We use formal methods of LTL progression to reward the produced logical forms by *progressing* each LTL logical form against the ground-truth trajectory, represented as a sequence of states, so that no LTL expressions are needed during training. We evaluate in two ways: on the SAIL dataset, a benchmark artificial environment of 3,266 trajectories and language commands as well as on 10 newly-collected real-world environments of roughly the same size. We show that our model correctly interprets natural language commands with 76.9% accuracy on average. We demonstrate the end-to-end process in real-time in simulation, starting with only a natural language instruction and an initial robot state, producing a logical form from the model trained with trajectories, and finding a trajectory that satisfies sequential constraints with an LTL planner in the environment.**

I. Introduction

A typical instruction following task is challenging for two reasons. First, the complex (e.g., temporal, sequential, conditional) constraints have to be represented in the form of logical goal specifications that state-of-the-art planners can take in. Assuming that these logical forms accurately represent the meaning of the natural language instruction, the planner has to temporally keep track of intermediate constraints while also reaching the final goal. The first problem can be resolved if we learn to parse natural language to more complex logical forms like Linear Temporal Logic (LTL) [8, 31] thus allowing the second planning problem to be solved by building LTL planners [18, 19, 31] that can find a path in the environment to satisfy the required temporal constraints. However, the semantic parsing problem in itself is extremely challenging, typically requiring paired data of language and LTL to train models to learn this translation function. This is data that is expensive to collect (in terms of time and money spent per task), difficult to collect (in terms of worker inaccuracies for this complex task) and moreover, models trained in this way are extremely brittle and fail to generalise to samples different from those seen during training [19, 28].

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

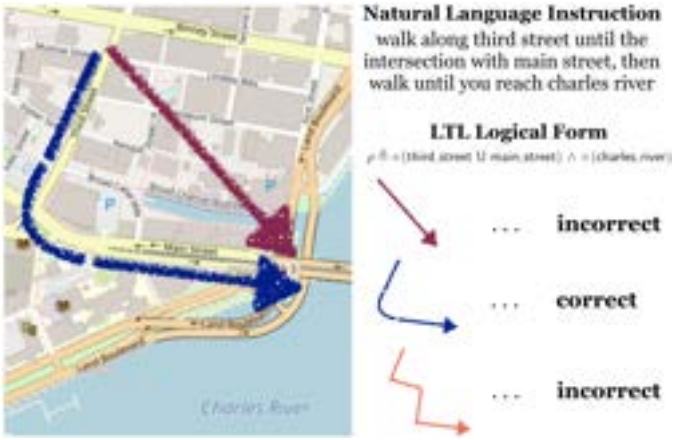

Fig. 1. Navigational instructions are often *path-oriented* and depend on intermediate states rather than just dependent on the final goal state. While all paths in the figure reach the goal, only the blue correctly follows the specified temporal constraints. Our model can learn to map English to LTL expressions that satisfy the constraints with no LTL at training time.

To address this, we propose to learn a semantic parsing model that does not require paired data of language and LTL logical forms, but instead learns from *trajectories* as a proxy. To collect trajectories on a large scale over a range of environments, we use path-finding algorithms to simulate trajectories in each environment. We then ask humans to annotate these trajectories with natural language instructions, creating datasets of language paired with trajectories in each environment. To validate the model-produced logical forms with trajectories, we use formal methods of LTL progression [8, 40], to check satisfiability of logical forms against ground-truth paths. Once the trained model can produce these grounded LTL representations, we then use a planner with LTL-based rewards to follow the instruction. Our framework enables a robot to learn to map between language and LTL expressions with no LTL annotations required during training; only a dataset of English and trajectories, as well as a model of the environment. We test our approach on a test set of unseen natural language instructions in each environment. We evaluate on our newly collected range of different environments with natural language annotations, as well as an existing benchmark dataset [32], using metrics for path and goal-state accuracy.

In this paper we present three main contributions. First, we ground natural language instructions to temporal logical form with no supervision of ground-truth logical forms. Second, our use of LTL as a meaning representation not

only allows handling of temporal ordering but also enables the use of formal methods of progression that allow a more efficient feedback mechanism than previous approaches. Third, we test the generalisability of this method in more than 10 different environments (both artificial and real-world) by annotating trajectores with natural language instructions in each environment. We release this data as well as the data-collection procedure to algorithmically simulate paths in large environments. We see the benefits of using a more expressive language such as LTL in instructions that require temporal ordering, and also see that the *path* taken with our approach more closely follows constraints specified in natural language. This dataset consists of 10 different environments, with up to 2,458 samples in each environment, giving us a total of 18,060 samples. To the best of our knowledge this is the largest dataset of temporal commands in existence.

II. Related Work

In the semantic parsing literature, previous work has explored weak supervision methods to ground natural language to lambda calculus expressions or SQL queries for retrieval tasks. Related to navigation, previous work [4, 26] has explored weakly supervised semantic parsing models. However, these ground to lambda calculus expressions and logical forms that do not handle temporal order in the way that LTL does. To the best of our knowledge, there is currently no work that attempts to ground natural language to temporal logic without supervision of logical forms during training.

LTL has been explored in the RL literature, to formulate tasks, either by creating reward functions that maximise the probability of satisfying the LTL formula [31, 41] or by guiding policy search with a measure of distance to satisfaction of the task [29]. Other work exploits the structure of LTL to decompose tasks into subtasks [40] to deal with temporal abstraction. Below, we enumerate the different avenues explored by previous instruction following methods.

a) Language → Logical Form: There is ample prior work on supervised semantic parsing to logical forms other than LTL [6, 39, 43, 44] as well as to LTL [19], however all of these require paired data of language and logical form to train models. Other weakly-supervised work learns semantic parsers without explicit annotation of logical forms, by allowing the execution of learned logical forms to act as supervision in varying domains e.g., conversational logs [3], system demonstrations [4, 13, 17, 42] and question-answer pairs [14, 30]. However, these are all unrelated to navigation and planning, and the weak supervision required is much simpler, usually requiring one execution (e.g., against a database) as opposed to a longer sequence of executions (e.g., a trajectory in an environment). Most relevant to our work is the model of Artzi and Zettlemoyer [4], however our work is different in that it grounds to LTL (rather than CCG expressions) and also provides a more efficient feedback mechanism making use of LTL progression.

b) Language → Plan: Also relevant is the body of work which seeks to map natural language directly to action sequences, e.g. [7, 15, 33, 34]. Such methods are typically trained end-to-end, and do not pass through an explicit intermediate logical form, as we do in this work. Having an intermediate logical form can help interpretability and ensure correctness, by first representing intended semantic meaning. It also helps generalisability in different domains as opposed to sequence-to-sequence models trained to produce paths in one environment that cannot generalise or exhibit compositionality [25, 28].

c) Logical Form (LTL) → Plan: Assuming we have correct logical goal specifications, there is ample work that explores how to plan and solve tasks *temporally* in environments. Previous work [8, 19, 31] has used planners in combination with LTL rewards to keep track of previous states. Recent approaches have used Deep Q-Networks with LTL specifications [40], by making use of LTL based rewards [31] where the input to the Q-value function is both the state and progressed LTL task. Other work uses hierarchical RL methods [27, 38] in the options framework, by creating one option per proposition with terminal states defined by states in which the proposition is true; giving the state and progressed LTL task as input to a meta-controller. While this line of work highlights the benefits of LTL for temporal abstraction, it does not deal with natural language, but focuses on task-solving when given the logical representations. It is therefore separate from the language grounding task that first attempts to convert natural language into representations that these methods can take in.

III. Linear Temporal Logic

In this section we explain the preliminaries of Linear Temporal Logic, the logical formalism into which we parse each natural language instruction. We go over the syntax and the semantics of the language and how they can be used in correspondence with states in an environment. We show how these semantics can be used for *progression* to check satisfiability of an LTL expression against a sequence of states (i.e., a trajectory) in the environment. Given that our weakly-supervised translation model does not have ground-truth LTL data to train on, the LTL progression is what supervises each logical form with a trajectory.

a) LTL Syntax: LTL has the following grammatical syntax:

$$\phi ::= \pi \mid \neg\phi \mid \phi \wedge \varphi \mid \phi \vee \varphi \mid \diamond \phi \mid \Box\phi \mid \bigcirc \phi \mid \phi \,\mathsf{U}\, \varphi$$

where the operators $\neg, \wedge, \vee$ are the logical connectives for *negation, and, or* and the temporal operators are $\diamond$ for *eventually*, $\Box$ for *globally*, U for *until* and $\bigcirc$ for *next*. Our set of propositions P consists of observable elements in the environment e.g., $(\mathsf{at\_object}, \mathsf{is\_intersection}, \mathsf{is\_corridor})$. All LTL expressions are constructed from the set P and the extended set of operators defined above i.e., the Boolean operators $\wedge, \vee, \neg$ and the temporal operators $\bigcirc, \mathsf{U}$. From these we can define $\Box$ (*always*) and $\diamond$ (*eventually*) for e.g., $\diamond\phi = \mathsf{true}\,\mathsf{U}\,\phi$.

b) LTL Semantics: Given the observable elements in the environment that form atomic propositions, the truth value of an LTL formula is determined relative to a sequence of tr assignments $\sigma =< \sigma_1, \sigma_2, \sigma_3, ... >$ where each state σ_i assi a value of true or false to each proposition.

A proposition $\rho \in \sigma_i$ indicates that the proposition ρ true in the state σ_i, thus allowing us to check existence propositions in states for progression functions in Table I.

c) LTL Progression: Given a sequence of truth assi ments and an LTL task specification, an LTL formula be *progressed* along the sequence. For example, the t $\diamond(p \wedge \bigcirc \diamond q)$ (i.e., eventually p and eventually q) can progressed to $\diamond q$ (i.e., eventually q) once the agent reac a state where p is true. Table I shows how we define progression functions.

TABLE I
SEMANTICS OF LTL PROGRESSION FUNCTIONS.

$\mathsf{prog}(\sigma_i, p)$	true if $p \in \sigma_i$, where $p \in P$
$\mathsf{prog}(\sigma_i, p)$	false if $p \notin \sigma_i$, where $p \in P$
$\mathsf{prog}(\sigma_i, \neg\phi)$	$\neg\mathsf{prog}(\sigma_i, \phi)$
$\mathsf{prog}(\sigma_i, \phi_1 \wedge \phi_2)$	$\mathsf{prog}(\sigma_i, \phi_1) \wedge \mathsf{prog}(\sigma_i, \phi_2)$
$\mathsf{prog}(\sigma_i, \phi_1 \vee \phi_2)$	$\mathsf{prog}(\sigma_i, \phi_1) \vee \mathsf{prog}(\sigma_i, \phi_2)$
$\mathsf{prog}(\sigma_i, \bigcirc\phi)$	ϕ
$\mathsf{prog}(\sigma_i, \Diamond\phi)$	true U ϕ
$\mathsf{prog}(\sigma_i, \phi_1 \,\mathsf{U}\, \phi_2)$	$\mathsf{prog}(\sigma_i, \phi_2) \vee (\mathsf{prog}(\sigma_i, \phi_1) \wedge \phi_1 \,\mathsf{U}\, \phi_2)$

IV. MODEL

In this section we describe the semantic parsing model that takes in natural language instructions and converts them to LTL expressions. Our model is based on previous work [16, 20] which trains semantic parsers from denotations by searching through a space of programs during training. However, applications of previous work were to ground language to SQL queries or logical operators to be executed against databases. We formulate this as a sequence prediction problem by representing an LTL program as a sequence of atomic propositions and operators in postfix notation. Each produced LTL expression is given a binary reward by progressing it along a ground-truth path. We explain the components of the model below.

a) Program Representation: Every token in an expression is either an operator of fixed arity (i.e., one or two arguments) or an atomic proposition. For example, the LTL expression `( a ∧ ( ◇ b ) )` converted to postfix notation `a b ◇ ∧` can be realised in an environment where the propositions correspond to locations. [1] This linearised representation allows easier execution with a stack based on the semantics of operators and propositions as shown in Figure 2. The vocabulary of atomic propositions is the set of all observable elements. In the SAIL environment, the simulated environment we use in Section V-A, this consists of e.g., *stool, brick corridor..*, while in every real world environment (see Section V-B) this

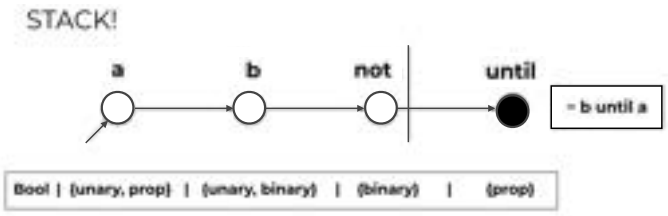

Fig. 2. Using a stack for type-constrained decoding to ensure syntax of produced logical forms.

b) Encoder and Decoder: Our training samples are of the form (x, t) where x is a natural language instruction and t is a trajectory in the environment. As in Guu et al. [20] our model generates program tokens $z_1, z_2..$ from left to right using a neural encoder-decoder model [37]. We encode every utterance x with a bidirectional LSTM [21] to create a contextualised representation h_i for every input token x_i. Our decoder is then a feed-forward network with attention [5] over the output from the encoder, that takes as input the last K decoded tokens. Formally, the probability of a decoded LTL expression is the product of the probability of its tokens conditioned on the history i.e., $p_\theta(z|x) = \prod_t p_\theta(z_t|x, z_{1:t-1})$ and the probability of a decoded token comes from the learned parameters and embedding matrices. We keep track of the *execution history* i.e., the k most recent tokens $z_{t-k:t-1}$ and concatenate their embeddings. While encoding, the input vocabulary consists of all the words contained in the natural language utterances, that are embedded with word vectors. While decoding, the output vocabulary is composed of all the propositional symbols (i.e., landmarks) in the environment as well as the operators *until, and, or, not, eventually, globally, next*. As shown in Figure 2 we use type-constrained decoding to ensure that the decoded programs are syntactically correct, while the LTL progression ensures that they are semantically correct, in that they give a reward of 1 against the ground truth trajectory. Using type-constrained decoding for the neural model ensures that the space of logical forms explored are always syntactically correct. Typical supervised methods would compare each produced token with the ground-truth logical form to ensure semantic correctness, however we do this in a weakly supervised way, by progressing the final logical form against the ground-truth trajectory in the environment.

c) Supervision: In a standard supervised setting, each produced logical form could simply be compared to the ground truth logical form in the data. In our weakly supervised approach, we instead use *trajectories* to supervise the produced logical forms during training. Each logical form gets a binary (1 or 0) reward by *progressing* it along the ground-truth trajectory, using the progression functions defined in Table I. Specifically, for an LTL task ϕ and state σ_i, we can update ϕ at each point in time i to reflect the parts of ϕ that have

[1] We show example conversions as well as example decodings of our model in attached supplementary material.

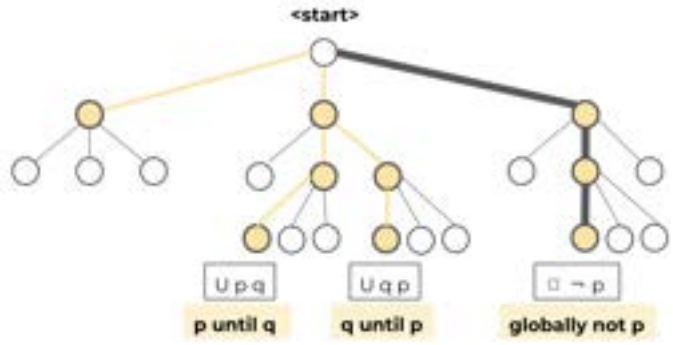

Fig. 3. Search algorithm to produce logical programs.

been satisfied. We can do this because if a sequence of truth assignments (i.e., a trajectory) satisfies an LTL formula at time i, then the formula progressed through ϕ_i is satisfied at time $i+1$. An exception to this is a global constraint e.g., *"always avoid A"* that requires the condition to always be met and cannot be progressed. We handle this by simply ensuring that the constraint holds at every time step.

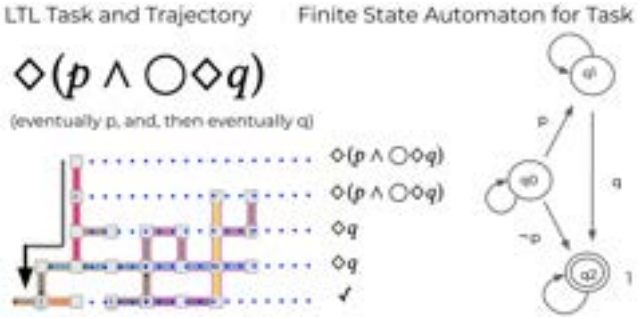

Fig. 4. Example of progression an ltl expression against a trajectory in the environment. Semantics are as defined in Table I.

A. Training

To train the model, we randomly initialise model parameters and optimise the objective function via stochastic gradient ascent, following the same training procedure in Guu et al. [20]. The token embedding size is 12, the beam size is 20 and the BiLSTM state dimensions are 30. The hidden state dimension of the decoder is 50, and this takes in the encoder representation, as well as the last 4 decoded tokens as input. We use Adam [24] as an optimiser with a learning rate of 0.001. We use a mini-batch size of 8 and train for 40k iterations to achieve the reported results. For a more detailed explanation of the search algorithm and policy gradient updates for the objective function, we refer readers to Guu et al. [20].

B. Planning in an Environment

Once we produce LTL interpretations of instructions, we use an LTL-based planner [] that takes in the LTL task specification to find a path in the environment. Crucially, this path not only reaches the final goal-state but also ensures that

Algorithm 1 Algorithm for Supervision with LTL Progression

```
Create training samples (u, t) of utterance and trajectory pairs
Create LSTM encoder, feed-forward decoder, search algorithm, reward function
Build environment representation with states composed of landmarks
for each sample (u, t) do
    e ← encoder(u) .. encode utterance with LSTM encoder
    program ← decoder(e) .. decode top k programs with feed-forward decoder for beam size k
    for each program p do
        for each state s in t do
            p', a ← prog(p) .. progress the LTL expression and return the updated expression p' and the current truth assignment a
        end for
        if a = True then
            reward ← 1
        else
            reward ← 0
        end if
        Update reward(p) for programs
    end for
end for
```

sequential constraints in the LTL formula are satisfied (e.g., first going to a certain location before reaching the goal). We use an MDP planner, adapted for LTL, as described below.

a) Markov Decision Processes: A *Markov Decision Process (MDP)* is a tuple $M = (S, A, T, R, \gamma)$ where S and A are a finite sets of *states* and *actions*, $T : S \times A \times S \to [0, 1]$ is the *transition function*, $R : S \times A \times S \to \Pr(R)$ is the *reward function* and γ is the *discount factor*. An agent learns a *policy* π i.e., a probability distribution over state-action pairs, that allows it to determine the actions it should take in each state with probability $\pi(a|s)$.

b) Linear Temporal Markov Decision Processes: To plan with temporal constraints, previous work combines the LTL expression with the environment MDP in order to make an expanded MDP that can keep track of the relevant parts of the LTL state. A labelling function therefore annotates the transitions with labels i.e., valid propositions for each state, thus allowing the checking of satisfiability of LTL expressions. These MDPs have previously been used for planning over an MDP to satisfy an LTL formula [36]. Abstract Product MDPs (AP-MDPs) [35] have been used to combine labelled MDPs that can handle temporal expressions along with ones that can solve tasks at different levels of state abstractions. This work does not deal with different levels of abstraction, therefore we only use a product MDP at the lowest level of abstraction i.e., the environment composed of individual grid cells.

Fig. 5. Examples of 4 different OSM maps in our data, with their corresponding number of landmarks available from the API.

V. Environments and Data

To compare with state of the art and existing semantic parsing models, we evaluate on SAIL [32] — a benchmark artifical environment. SAIL is an existing dataset with 3,000 samples of instructions and trajectories.

While the language used is considerably complex, SAIL contains very little temporal language (e.g., use of words like *until* or multiple sequential constraints or landmarks to be met). To properly test handling of temporal language, as well as generalisability to real-world, environments, we also test on language commands in 10 newly-collected real-word OSM environments in different cities in the US. In all of these, our model trains without supervision of ground truth logical forms, requiring instead only example trajectories in the environment.

Here, we explain both environments and their complexities. Section V-A describes the artificial environment while Sections V-B, V-B2, V-B3 and V-B4 describe the real-world environments, as well as the methods we use to algorithmically generate thousands of trajectories in OSM environments to create large-scale datasets for training.

A. SAIL: A benchmark dataset for instruction following

SAIL, introduced in MacMahon et al. [32], is a navigation dataset containing route instructions annotated with trajectories for three different environments of varying size. An environment is composed of connected hallways with different floor patterns (grass, brick, wood, gravel, blue, flower, or yellow octagons), wall paintings (butterfly, fish, or Eiffel Tower) and objects (hat rack, lamp, chair, sofa, barstool, and easel) at intersections. In SAIL, the challenge of learning to ground natural language stems from the fact that instructions given by humans are complex, free-form and of variable length (either 3,266 single sentences in isolation or 706 full paragraphs). While this task and dataset bear superficial similarity to others [1, 9], the language and paths required here are quite different — the proportion of instructions to actions is much higher, the interpretation of language is highly compositional and instruction length varies widely. SAIL has therefore been the subject of focused attention in semantic parsing, resulting in a range of different approaches that attempt to plan in such settings.

B. Open Street Maps: Instruction following in real world environments

1) Open Street Maps (OSM): We use Open Street Maps (OSM), a global open-sourced map API where users can add landmarks, as well as information about the landmarks that are then verified. This therefore gives us access to real world maps, names of landmarks and cartesian coordinates of their locations. We query this semantic database to get maps for 10 cities across the US.

We chose cities across the US, in areas that have open spaces and interesting landmarks or statues that can provide referring expressions, e.g., around universities and parks. We provide detailed examples, lists of cities, and landmarks within cities in the supplementary material. Figure 5 shows examples of pictures of the map in 4 different possible environments.

2) Converting Real World Maps to Underlying Graph Structure: In order to convert dense OSM maps composed of (latitude, longitude) pairs of points into a graphical structure in which we can simulate paths, we use Voronoi cells [] to partition the map. Given an OSM map composed of landmarks in a 300m radius square, the map can be partitioned into Voronoi cells as shown in Figure 6. This is done (as in the original Voronoi implementation) by randomly generating points inside the bounded region. Triangulation is done by connecting each node to it's nearest neighbours, forming a network of triangles. The boundaries of Voronoi cells are then formed by connecting perpendicular bisectors of triangles. [] shows that Voronoi-based enables faster and more efficient planning over larger distances.

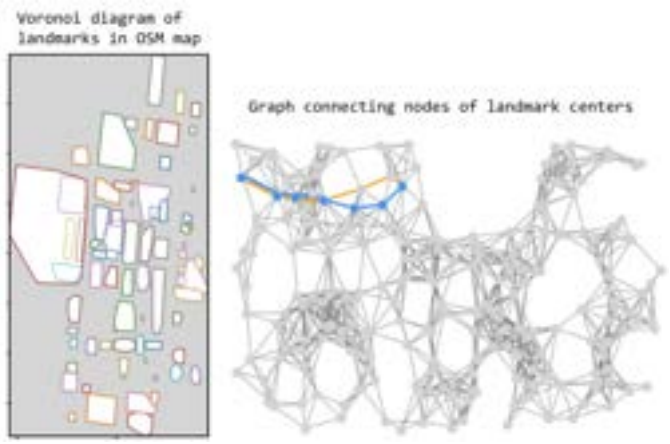

Fig. 6. Creating Voronoi diagrams and underlying connecting graphs of the environment.

3) Algorithmic Simulation of Trajectories in Environments: Most previous work (that is not concerned with large-scale tra-

jectory data collection) hand-annotates trajectories in graphs. This collection of trajectories is then given to a human for an annotation task, thus obtaining paired (language, trajectory) data in an environment. When dealing with multiple large environments, with thousands of trajectories per environment, manually annotating trajectories quickly becomes infeasible.

Therefore, we instead *simulate* trajectories in the different environments using path-sampling algorithms. Specifically, given our environment graph that connects landmark nodes to one another, we sample start and end nodes (s, e) from a uniform distribution. We use a k-th shortest path sampling algorithm (where k is sampled from a geometric distribution) to obtain a path between between nodes s and e. It is crucial to note that the paths obtained are therefore not always the shortest path between two nodes — a bias that can be quickly learned by models, that has been shown [22] to make them ignore important components of the input (e.g., if agents are trained on paths that are always the direct shortest path between two nodes, there is no need for them to condition on the natural language instruction, environment features and so on). In this way, we obtain up to 3,000 algorithmically simulated trajectories for each environment, distributed over different start and end nodes. We ensure that the lengths of the paths vary (i.e., are not all the same length) and that all landmarks in the graph have been covered in a subset of the trajectories.

Fig. 7. Algorithmic path simulation.

4) Collecting Natural Language Annotations for Environments: Once we can algorithmically simulate thousands of trajectories in different environments, we collect natural language instruction data for trajectories using Amazon Mechanical Turk (AMT). Each AMT task consists of a picture of the real-world OSM map, as well as two trajectories drawn in different colours (blue and orange) on the graph of connecting landmarks. The two different trajectories are obtained from sampling from the k-th shortest paths, and we instruct workers to give a natural language instruction that describes *one* of the paths while specifically excluding the other as a possibility. This therefore elicits language instructions specific to one of the trajectories, ensuring that they refer to elements along the target path. Moreover, sampling from the k-th shortest trajectories ensures that the paths in our data are not always shortest paths from the start to the end, a heuristic that neural models have been shown to easily game [28]. We also provide a map key with the list of landmarks in the map, to make the task easier for workers. For each trajectory, we collect 3 different natural language commands, thus giving us an average of 2,884 trajectories per environment, for 10 different environments. To the best of our knowledge, this is the largest collection of temporal data aimed at LTL, for real-world environments.

Algorithm 2 Algorithmically simulating trajectories

```
Initialise number of paths required
for each OSM map M do
    V ← Voronoi(M) .. voronoi diagram of landmarks
    G ← Landmarks(V) .. graph with nodes as landmark centers, edges as distances between landmarks
    for i in num_paths do
        s ← random(G.nodes) .. randomly sample a start node for path
        e ← random(G.nodes) .. randomly sample an end node for path
        k ← random(geometric(p=0.2) .. sample k from geometric distribution
        paths ← dijkstra(s, e) .. find paths between s and e from shortest to longest
        path_1 ← paths[k] .. choose the k^th shortest path as the ground truth path
        path_2 ← random(paths) .. randomly select an alternative path different from the above
    end for
end for
```

VI. Experimental Evaluation

In this section we explain the environments and evaluation metrics, that attempt to evaluate not only the final goal location, but the entire path. To compare to previous instruction-following models on the benchmark set, we report goal-state accuracy on SAIL, but also propose to evaluate path metrics, to ensure that the entire trajectories are correct. We compare to all past models that evaluated on SAIL, as reported for the test set. To evaluate the real-world OSM environments, we report goal-state accuracy as well as path accuracy, comparing to a baseline that does not use LTL.

A. Evaluation Metrics

To evaluate models on the SAIL dataset, previous works compare the agent's end state to a labelled state s' i.e., the end point of the ground-truth trajectory, for all (single and multi-sentence) instructions in the test set. As explained in this section, we propose additional evaluation metrics to compare the entire path rather than just the goal location.

To directly compare our approach with existing work, we measure the *goal-state* accuracy i.e., the ability of the model to reach the same location in the environment as the ground-truth trajectory. Unlike prior work, we also propose a more fine-grained analysis to evaluate *path accuracy*. Often times, the path taken to reach the final goal location is crucial – especially when the instruction specifies constraints on how to reach the

goal. A more specific analysis of the entire path is even more important in complicated environments with several possible paths that reach the goal. Metrics that only measure success rate (i.e., reaching the final goal location) and disregard the path taken will therefore not distinguish between such paths. We therefore propose to evaluate correctness of the produced paths in comparison to the ground-truth path. To evaluate the sequences, we treat paths as vectors of indices in the grid world and compute precision, recall, accuracy and edit distance to the gold path. Paths that more closely follow the required constraints will therefore have higher precision, recall and accuracy, and a lower edit distance.

VII. Results

a) Comparison to prior language grounding work in SAIL: We compare our model to existing approaches that report final goal-state accuracy on the SAIL dataset. Most similar to our approach is the model from [4] that trains a CCG semantic parser supervised by trajectories. Other methods include algorithms supervised with logical forms [11, 13] that learn semantic parsers for instructions as well as ones that involve strategies for online learning of lexicons [10] and ones that use contextual information [12] for better language understanding. We also compare to the supervised alignment-based models [2] that build grounding graph representations to execute instructions and neural sequence-to-sequence models [33] that translate language to actions in the environment.

TABLE II
Evaluation of systems on the SAIL dataset.

System	Single	Multi
Chen and Mooney [11]	54.4	16.18
Chen [10]	57.28	19.18
+ additional data	57.62	20.64
Kim and Mooney [23]	57.22	20.17
Artzi and Zettlemoyer [4]	65.28	**31.93**
Andreas and Klein [2]	59.60	-
Mei et al. [33]	**71.05**	30.34
Ours	66.92	20.17

b) Performance in SAIL: Table II shows goal-state accuracy of systems on SAIL, while Table III shows evaluation metrics for paths produced, compared to ground-truth paths in the dataset. Table II compares to other work that is different in the form of supervision provided and model architecture used, but evaluates on the same end-task i.e., goal-state accuracy over the SAIL dataset. We see that our model has comparable performance to previous models in terms of navigating to the correct goal location. However, this metric can still reward incorrect paths (e.g., ones that violate specified constraints) as long as they end up in the correct final location. In Table III we compare to the best-performing model in terms of path accuracy. We see that our model outperforms the previous best-performing model when we evaluate the entire path taken. We furthermore analyse the *type* of natural language instructions that involve complex temporal constraints. We do this by taking all instructions in the dataset that contain natural language counterparts of temporal operators (e.g., *until, eventually, finally, always..*) and evaluate models specifically on this subset. These temporal sentences form 20% of the data (476) sentences. We see that our model that grounds to temporal logical form outperforms previous models under this finer-grained evaluation.

TABLE III
For path evaluation, we evaluate precision, recall, accuracy (higher is better) and edit distance (lower is better).

Data	System	Prec.	Recl.	Acc.	ED
All (SAIL)	**Seq2seq**	31.6	30.33	91.5	3.25
	Ours	**33.5**	**32.34**	**93.7**	**2.24**
		+1.9	+2.01	+2.2	-1.01
Temporal (SAIL)	**Seq2seq**	31.2	30.19	91.2	3.91
	Ours	**34.3**	**35.2**	**94.5**	**1.27**
		+3.1	+5.01	+3.3	-2.64

c) Comparison to baseline in real-world OSM environments: We now turn to the real-world environments with the newly-collected complex language commands. To evaluate the generalisability of our approach in multiple real-world environments with complex langauge, we evaluate performance in each of the new environments, after grounding natural language instructions to logical forms, and then planning with logical forms in each environment. We compare to a baseline that does not use LTL. This model takes in a natural language utterance and predicts a final goal location from the set of elements in the environment; therefore not considering all the sequential constraints. The planner then finds a path from the start location to the goal. This model is a strong competitor in instances that do not require temporal reasoning (e.g., to directly go to one goal location without needing to meet path constraints), however for complex, sequential tasks, this does not account for temporal ordering and intermediate tasks. We use a Multi-Layer Perceptron (MLP) classifier, that encodes the natural language instruction and predicts the final goal-location (i.e., landmark) from the set of all landmarks. Each training sample of language and trajectory pairs therefore gives us a training sample of language paired with the correct end-goal location, which is obtained by simply taking the last point of the ground-truth trajectory. We use a cross-entropy loss against the correct goal-state and train this model for each environment. A model that correctly predicts the goal can therefore achieve perfect goal-state accuracy when this is given to a planner in the environment. However, this does not give us any guarantees on the *path* taken. This baseline model therefore has the same number of training samples as the weakly supervised LTL parser, but performs a simple prediction task, rather than a semantic parsing task composed

TABLE IV
EVALUATION ON ALL ENVIRONMENTS. THE FIRST NUMBER SHOWS GOAL-STATE ACCURACY (HIGHER IS BETTER) WHILE THE SECOND SHOWS EDIT DISTANCE COMPARED TO THE GROUND-TRUTH TRAJECTORY (LOWER IS BETTER).

	Austin, TX	**Ann Arbor, MI**	**Atlanta, GA**	**Baltimore, MD**	**Berkeley, CA**	**SAIL**
Ours	89.4 / 1.1	84.3 / 1.9	77.6 / 1.6	74.5 / 1.4	80.3 / 1.5	66.9 / 2.1
Baseline	66.7 / 3.4	78.4 / 3.0	74.9 / 3.2	68.3 / 3.5	70.3 / 4.3	74 / 2.2
	Boston, MA	**Cambridge, MA**	**New Haven, CT**	**Philadelphia, PA**	**Providence, RI**	**SAIL (T)**
Ours	69.8 / 3.1	68.8 / 2.2	88.5 / 1.1	89.9 / 1.8	76.6 / 1.2	74.3 / 1.1
Baseline	60.3 / 3.3	52.3 / 3.4	76.5 / 2.6	63.5 / 3.9	74 / 2.2	70.1 / 4.3

of operators and operands.

d) Performance in OSM: Table IV shows goal-state accuracy as well as path metrics in each OSM map. Each test set is composed of a 100 different unseen trajectories with natural language commands of varying length. We compute goal-state accuracy by evaluating whether or not the final location after planning is the correct end location of the trajectory. We compute path accuracy by computing the edit distance between the computed path and the ground-truth trajectory. We report the average over all samples in each test set. We see that our model outperforms the non-LTL baseline on the path metric of edit distance (lower is better), since reasoning temporally enforces meeting all the intermediate constraints, therefore allowing the planner to find a path that matches the ground-truth path. We also see that goal-state accuracy is higher. This is because there exist several cases where the ordering of referents in the natural language statements can be reversed (e.g., both sentences *"go to B and then finally to A" "go to A after going to B"* have the same meaning, but in the latter, the location *A* is referred to before the location *B* in the natural language sentence, thus causing the final goal location to be predicted incorrectly).

e) Demonstration in Simulation: To demonstrate the working of our entire pipeline in real time, we demonstrate the execution of natural language instructions in simulation. This can then be connected to the Skydio R1 drone, that navigates over actual landmark coordinates in any of the real-world environments. Due to several drone navigation restrictions, we demonstrate this only inside the simulator, however navigating over the actual real-world environment coordinates.

Fig. 8. Example still from our robot demonstration in simulation with a Skydio drone.

Specifically, for our demonstration video, we sample a different human-given natural language instruction (that was never used during training) in each of the 10 environments. In real-time, we parse these with the weakly-supervised model trained in the environment, plan with the produced LTL specification, and show the drone navigating over the planned path. A video of the demonstration attached with this paper, shows the Skydio drone in simulation following these commands in different environments, demonstrating the entire end-to-end process. All of these environments follow our weakly supervised approach described above. This means, that this demonstration and real-time usage can be applied to any new environment in which we collect natural language annotations for trajectories, therefore allowing the model to train and then produce LTL representations that the planner needs.

VIII. CONCLUSION

In this paper we use a weakly supervised semantic parsing model to ground natural language to temporal logical form — a formal language that allows handling of complex, temporal events that are typically unable to be handled by most (traditionally Markovian) methods. We evaluate on both artificial (SAIL) and real-world (OSM) environments. While the SAIL navigation dataset was not specifically constructed with these temporal, sequential constraints in mind, the fine-grained evaluations show our method allows dealing with these naturally occurring constraints better than previous state-of-the-art methods. Moreover, in our newly collected OSM dataset that contains more complex, temporal language, we show that our method that can deal with temporal order, has superior performance. Future work involves incorporating the structure of LTL to jointly learn logical forms and execution models, especially in domains that specifically require temporal reasoning. Future work on the planning side involves working in partially observable domains, as well as dealing with landmarks and locations that were never seen before. Future work on the neural decoding side involves grounding to new elements in the environment that did not exist in the model's vocabulary during training. This will allow generalising to environments different from what was trained on (e.g., a new city) to allow planners to take in logical forms decoded in those environments. Future work can also explore more action-oriented language, rather than goal-based approaches,

as well as attempting to resolve spatial language commands to elements in the environment for better navigation.

References

[1] Anne H Anderson, Miles Bader, Ellen Gurman Bard, Elizabeth Boyle, Gwyneth Doherty, Simon Garrod, Stephen Isard, Jacqueline Kowtko, Jan McAllister, Jim Miller, et al. The hcrc map task corpus. *Language and speech*, 34(4):351–366, 1991.

[2] Jacob Andreas and Dan Klein. Alignment-based compositional semantics for instruction following. *arXiv preprint arXiv:1508.06491*, 2015.

[3] Yoav Artzi and Luke Zettlemoyer. Bootstrapping semantic parsers from conversations. In *Proceedings of the conference on empirical methods in natural language processing*, pages 421–432. Association for Computational Linguistics, 2011.

[4] Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62, 2013.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[6] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, 2013.

[7] Satchuthananthavale RK Branavan, Harr Chen, Luke S Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 82–90. Association for Computational Linguistics, 2009.

[8] J Richard Büchi. On a decision method in restricted second order arithmetic. In *The Collected Works of J. Richard Büchi*, pages 425–435. Springer, 1990.

[9] Guido Bugmann, Stanislao Lauria, Theocharis Kyriacou, Ewan Klein, Johan Bos, and Kenny Coventry. Using verbal instructions for route learning: Instruction analysis. *Proc. TIMR*, 1:96103, 2001.

[10] David L Chen. Fast online lexicon learning for grounded language acquisition. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 430–439. Association for Computational Linguistics, 2012.

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

[12] David L Chen, Joohyun Kim, and Raymond J Mooney. Training a multilingual sportscaster: Using perceptual context to learn language. *Journal of Artificial Intelligence Research*, 37:397–435, 2010.

[13] Henry Chen, Austin S Lee, Mark Swift, and John C Tang. 3d collaboration method over hololens and skype end points. In *Proceedings of the 3rd International Workshop on Immersive Media Experiences*, pages 27–30. ACM, 2015.

[14] James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. Driving semantic parsing from the world's response. In *Proceedings of the fourteenth conference on computational natural language learning*, pages 18–27. Association for Computational Linguistics, 2010.

[15] Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. Speaker-follower models for vision-and-language navigation. In *Advances in Neural Information Processing Systems*, pages 3318–3329, 2018.

[16] Omer Goldman, Veronica Latcinnik, Udi Naveh, Amir Globerson, and Jonathan Berant. Weakly-supervised semantic parsing with abstract examples. *CoRR*, abs/1711.05240, 2017. URL http://arxiv.org/abs/1711.05240.

[17] Dan Goldwasser and Dan Roth. Learning from natural instructions. *Machine learning*, 94(2):205–232, 2014.

[18] Nakul Gopalan, Marie desJardins, Michael L Littman, James MacGlashan, Shawn Squire, Stefanie Tellex, John Winder, and Lawson LS Wong. Planning with abstract markov decision processes. In *ICAPS*, 2017.

[19] Nakul Gopalan, Dilip Arumugam, LL Wong, and Stefanie Tellex. Sequence-to-sequence language grounding of non-markovian task specifications. In *Robotics: Science and Systems*, 2018.

[20] Kelvin Guu, Panupong Pasupat, Evan Zheran Liu, and Percy Liang. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. *CoRR*, abs/1704.07926, 2017. URL http://arxiv.org/abs/1704.07926.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[22] Vihan Jain, Gabriel Magalhaes, Alex Ku, Ashish Vaswani, Eugene Ie, and Jason Baldridge. Stay on the path: Instruction fidelity in vision-and-language navigation. *arXiv preprint arXiv:1905.12255*, 2019.

[23] Joohyun Kim and Raymond J Mooney. Unsupervised pcfg induction for grounded language learning with highly ambiguous supervision. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 433–444. Association for Computational Linguistics, 2012.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Philipp Koehn and Rebecca Knowles. Six chal-

lenges for neural machine translation. *arXiv preprint arXiv:1706.03872*, 2017.

[26] Jayant Krishnamurthy and Tom M Mitchell. Weakly supervised training of semantic parsers. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 754–765. Association for Computational Linguistics, 2012.

[27] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

[28] Brenden M Lake and Marco Baroni. Still not systematic after all these years: On the compositional skills of sequence-to-sequence recurrent networks. *arXiv preprint arXiv:1711.00350*, 2017.

[29] Xiao Li, Cristian-Ioan Vasile, and Calin Belta. Reinforcement learning with temporal logic rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3834–3839. IEEE, 2017.

[30] Percy Liang, Michael I Jordan, and Dan Klein. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446, 2013.

[31] Michael L Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. Environment-independent task specifications via gltl. *arXiv preprint arXiv:1704.04341*, 2017.

[32] Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. *Def*, 2(6):4, 2006.

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

[34] Dipendra Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning. *arXiv preprint arXiv:1704.08795*, 2017.

[35] Yoonseon Oh, Roma Patel, Thao Nguyen, Baichuan Huang, Ellie Pavlick, and Stefanie Tellex. Planning with state abstractions for non-markovian task specifications. *arXiv preprint arXiv:1905.12096*, 2019.

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

[37] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[38] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[39] Lappoon R Tang and Raymond J Mooney. Automated construction of database interfaces: Integrating statistical and relational learning for semantic parsing. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 133–141. Association for Computational Linguistics, 2000.

[40] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Teaching multiple tasks to an rl agent using ltl. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 452–461. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[41] Min Wen, Ivan Papusha, and Ufuk Topcu. Learning from demonstrations with high-level side information. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017.

[42] Edward C Williams, Nakul Gopalan, Mine Rhee, and Stefanie Tellex. Learning to parse natural language to grounded reward functions with weak supervision. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7. IEEE, 2018.

[43] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

[44] Luke S Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420*, 2012.

IX. Supplementary Material

Environment	Natural Language Instruction	LTL Task Specification
SAIL (Jelly)	Go forward one segment to the pink-flowered carpet hall	$\triangleq \bigcirc \mathtt{a}$ $\mathsf{a} = \mathsf{at\_floor\_flowered}$
SAIL (Grid)	Go forward along the pink-flowered carpet hall three segments to the blue-tiled hall, passing a hatrack maybe.	$\triangleq (\mathtt{a} \wedge \Diamond \mathtt{b})\, U\, (\Diamond \mathtt{c})$ $\mathsf{a} = \mathsf{at\_floor\_pink}, \mathsf{b} = \mathsf{at\_object\_hatrack},$ $\mathsf{c} = \mathsf{at\_floor\_blue}$
SAIL (L)	Follow the yellow path towards the chair	$\triangleq \mathtt{a}\ U \Diamond \mathtt{b}$ $\mathsf{a} = \mathsf{at\_floor\_yellow}, \mathsf{b} = \mathsf{at\_chair}$
OSM (Austin, TX)	Starting from Winged Victory go to Mustangs then finally go to Monochrome for Austin.	$\triangleq \mathtt{a}\ U\ (\Diamond \mathtt{b}\ U\ (\Diamond \mathtt{c}))$ $\mathsf{a} = \mathsf{at\_winged\_victory}, \mathsf{b} = \mathsf{at\_mustangs}$ $\mathsf{c} = \mathsf{at\_monochrome}$
OSM (Ann Arbor, MI)	Begin at Shapiro Library. From there, turn southwest and go until you reach Good Time Charley's.	$\triangleq \mathtt{a}\ U \Diamond \mathtt{b}$ $\mathsf{a} = \mathsf{at\_shapiro\_library}, \mathsf{b} = \mathsf{at\_charleys}$
OSM (Atlanta, GA)	Begin at the Student Center and go to Rotatious before ending at Panda Express	$\triangleq \mathtt{a}\ U\ (\Diamond \mathtt{b}\ U\ (\Diamond \mathtt{c}))$ $\mathsf{a} = \mathsf{at\_student\_center}, \mathsf{b} = \mathsf{at\_rotatious}$ $\mathsf{a} = \mathsf{at\_panda\_express}$
OSM (Baltimore, MD)	Walk north from William Welch to BMA Sculpture Gardens.	$\triangleq \mathtt{a}\ U \Diamond \mathtt{b}$ $\mathsf{a} = \mathsf{at\_william_welch}, \mathsf{b} = \mathsf{at\_bma}$
OSM (Berkeley, CA)	From Observatory Hill, go northwest to Memorial Pool.	$\triangleq \mathtt{a}\ U \Diamond \mathtt{b}$ $\mathsf{a} = \mathsf{at\_obs\_hill}, \mathsf{b} = \mathsf{at\_memorial\_pool}$
OSM (Boston, MA)	Head north, straight northeast slightly. When you get to the Fogg museum, make a left and head west and southward for a long time. Eventually you will come to Cafe Pamplona, that's your end point.	$\triangleq \Diamond \mathtt{a}$ $\mathsf{a} = \mathsf{at\_fogg\_museum}$
OSM (Cambridge, MA)	Start at Cambridge Bicycle and go straight to the MIT Museum. Then veer a little to the right to the Middlesex Lounge. Keep veering right to end up at The Asgard.	$\triangleq \mathtt{a}\ U\ (\Diamond \mathtt{b}\ U\ (\Diamond \mathtt{c}\ U\ (\Diamond \mathtt{d})))$ $\mathsf{a} = \mathsf{at\_cambridge\_bicycle}, \mathsf{b} = \mathsf{at\_museum}$ $\mathsf{c} = \mathsf{at\_middlesex}, \mathsf{d} = \mathsf{at\_asgard}$
OSM (New Haven, CT)	Head south towards Harkness Tower.	$\triangleq \Diamond \mathtt{a}$ $\mathsf{a} = \mathsf{at\_harkness\_tower}$
OSM (Philadelphia, PA)	Starting at Philadelphia Runners, travel east-southeast until you reach Annenberg Center.	$\triangleq \mathtt{a}\ U \Diamond \mathtt{b}$ $\mathsf{a} = \mathsf{at\_phil\_runners}, \mathsf{b} = \mathsf{at\_annen\_center}$
OSM (Providence, RI)	You will be starting at the Blue Room. From the Blue Room, you will walk Southwest for a while until you reach the Slavery Memorial. Once you have arrived there, you are at your destination.	$\triangleq \mathtt{a}\ U \Diamond \mathtt{b}$ $\mathsf{a} = \mathsf{at\_blue\_room}, \mathsf{b} = \mathsf{at\_slavery\_memorial}$

TABLE V
EXAMPLE COMMANDS FROM THE DIFFERENT DATASETS AS WELL AS LTL EXPRESSIONS PRODUCED BY THE MODEL.

LTL Task Specification	Postfix Form
$\triangleq \Diamond(a)$	$\triangleq \Diamond a$
$\triangleq (a\ U\ b)$	$\triangleq U\ a\ b$
$\triangleq (a \wedge b)$	$\triangleq \wedge\ a\ b$
$\triangleq (a \vee b)$	$\triangleq \vee\ a\ b$
$\triangleq (a \wedge b)\ U\ (c)$	$\triangleq U \wedge a\ b\ c$
$\triangleq (a \wedge b)\ U \Diamond (c)$	$\triangleq U \wedge a\ b \Diamond c$

TABLE VI

TABLE SHOWS EXAMPLE LOGICAL FORMS IN POSTFIX NOTATION, AS DECODED BY OUR MODEL. THE LETTERS A, B.. REFER TO PROPOSITIONS (E.G., A LANDMARK IN A MAP) WHILE THE SYMBOLS $\Diamond$, U, $\wedge$ REFER TO UNARY OR BINARY OPERATORS FOR *eventually, until, and* AND SO ON AS SPECIFIED IN TABLE **??**.