Learning to Ground Language to Temporal Logical Form

Anonymous EMNLP-IJCNLP submission

Abstract

Often times, natural language commands exhibit sequential constraints, e.g. *"go through the kitchen and then into the living room."* Conventional methods in classical Reinforcement Learning (RL) assume Markovian reward functions that cannot handle such non-Markovian constraints. We therefore propose to ground natural language commands to Linear Temporal Logic (LTL), but propose to do this when direct supervision is not available — utterances are labelled with trajectories but not the logical form itself. We use formal methods of LTL progression and model checking, to reward the learned latent logical forms that execute correct trajectories. These logical forms can then serve as input to planners that correctly solve the navigation task. We evaluate our framework on both goal state and path accuracy, as well as the ability to handle temporal language.

1 Introduction

In navigation and instruction-following tasks, an agent executes a series of actions to perform a task, in response to some task specification that defines the goal that the agent must reach. This is challenging for two reasons — first, the broad spectrum of possible language used must be grounded to a symbolic, logical representation, that can then be given to a model to find a policy that solves the task. Second,

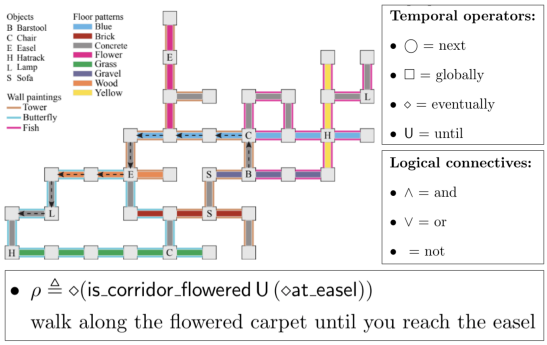

Figure 1: Example path and instruction grounded to temporal logical form for one of the environments from (MacMahon et al., 2006). Letters indicate object positions, while colours indicate floor patterns and wall paintings, that all form propositions for LTL representations. Agents are required to correctly interpret the meaning of the instruction to navigate to the intended goal location.

the complex (e.g., temporal, sequential, conditional) constraints are non-trivial to express and achieve by conventional methods in classical Reinforcement Learning (RL), that assume a Markovian reward function. For example, commands that specify temporal ordering of events such as *"only enter the living room after you have been to the kitchen"* could potentially map to unbounded action sequences, unless we have adequate representation of state properties in accordance with temporal patterns over time.

We propose to ground natural language instructions to LTL task specifications that can

handle temporal order and furthermore allow the logical formula to be structured and decomposed into paths in an environment. Moreover, we attempt to do this by learning from demonstrations. While collecting and annotating symbolic logical forms is an expensive process, we can more easily collect *executions* of these logical forms as trajectories in our environment. Our system takes in natural language as input, grounds to symbolic temporal logical form, and checks this against a human-annotated output trajectory as a form of weak supervision. Our work is novel in that we ground to LTL logical form (that can handle temporal constraints) and moreover allows using methods like LTL progression, to check satisfiability against trajectories in order to reward the learned latent LTL forms. In comparison to previous approaches that build execution models or planners in order to reward logical forms, our approach is an order of magnitude more efficient in terms of complexity, given that we only need to progress an LTL expression against a fixed-length trajectory, rather than execute an expression and solve a planning problem each time. We evaluate our method on a benchmark dataset and highlight the benefits of using a more expressive language, such as LTL, that can handle temporal order.

2 Related Work

Previous work has used LTL to formulate tasks in RL, either by creating reward functions that maximise the probability of satisfying the LTL formula (Wen et al., 2017; Littman et al., 2017) or by guiding policy search with a measure of distance to satisfaction of the task (Li et al., 2017). Other work exploits the structure of LTL to decompose a task into subtasks (Toro Icarte et al., 2018) to deal with temporal abstraction. To the best of our knowledge, there is currently no work that attempts to ground tasks given in natural language to temporal logic without annotation of logical forms, to exploit LTL structure for further planning.

Language → Logical Form There is ample prior work on supervised semantic parsing to LTL (Gopalan et al., 2018) and to logical forms other than LTL (Zettlemoyer and Collins, 2012; Tang and Mooney, 2000; Berant et al., 2013; Yu et al., 2018). Relevant to our work is past work which learns semantic parsers without explicit annotation of logical forms, by allowing the execution of the learned logical form to act as supervision e.g., conversational logs (Artzi and Zettlemoyer, 2011), system demonstrations (Chen et al., 2015; Goldwasser and Roth, 2014; Artzi and Zettlemoyer, 2013; Williams et al., 2018) and question-answer pairs (Clarke et al., 2010; Liang et al., 2013).

Language → Plan Also relevant is the body of recent work which seeks to map natural language directly to action sequences, e.g. Mei et al. (2016); Misra and Artzi (2015); Branavan et al. (2009); Misra et al. (2017); Fried et al. (2018). Such methods are typically trained end-to-end, and do not pass through an explicit intermediate logical form, as we do in this work.

Logical Form (LTL) → Plan (Dzifcak et al., 2009; Gopalan et al., 2018) explore grounding language to LTL and then planning with an existing planner, given LTL task specifications. However (Gopalan et al., 2018) use a standard sequence-to-sequence model trained on annotated utterances, and they note that this method fails to exhibit compositionality or generalise to unseen logical forms. Several recent approaches use Deep Q-Networks with LTL specifications (Toro Icarte et al., 2018), by making using of LTL based rewards (Littman et al., 2017) where the input to the Q-value function is both the state and the progressed LTL

task. Other work uses hierarchical RL methods (Sutton et al., 1999; Kulkarni et al., 2016) 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.

3 Grounding Language to Linear Temporal Logic with Trajectories

We propose to ground natural language to temporal logical form, where grounded LTL expressions are only supervised by trajectories in the environment. Our model therefore receives natural language instructions (as a sequence of words) as input and is supervised by an executed path (as a sequence of points specifying (x, y) locations) in the environment. Once the natural language instruction has been grounded to an LTL expression, this can be used to plan in a given environment using existing methods such as Q-learning with LTL-based rewards. This entire pipeline therefore ensures that the intended meaning of the instruction is first represented in logical form (that satisfies temporal constraints) and then used to navigate in the environment (to reach the final goal location). In this section we give an overview of the different components that our approach relies on.

3.1 Linear Temporal Logic

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*. We can also define the symbols true and false through the following equivalences: $\mathsf{true} \equiv \phi \vee \neg\phi$ and $\mathsf{false} \equiv \neg\,\mathsf{true}$. Our set of propositions P consists of observable elements in the environment that trajectories can pass through e.g $(\mathsf{at\_object}, \mathsf{is\_intersection}, \mathsf{is\_corridor})$. All LTL expressions are constructed from the set of propositional symbols 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., $\diamond\,\phi = \mathsf{true}\,\mathsf{U}\,\phi$.

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 truth assignments $\sigma =< \sigma_1, \sigma_2, \sigma_3, ... >$ where each state σ_i is an assignment of true or false values to propositions. A proposition $\rho \in \sigma_i$ indicates that the proposition ρ is true in the state σ_i.

LTL Progression: Given a sequence of truth assignments and an LTL task specification, an LTL formula can be *progressed* along the sequence. For example, the task $\diamond(p \wedge \bigcirc \diamond\, q)$ (i.e., eventually p and eventually q) can be progressed to $\diamond q$ (i.e., eventually q) once the agent reaches a state where p is true. In an RL setting, the LTL formula can be updated to reflect the agent's actions and the parts of the formula that have been satisfied so far.

$\mathsf{prog}(\sigma_i, p)$	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
$\mathsf{prog}(\sigma_i, \phi_1\,\mathsf{U}\,\phi_2)$	$\mathsf{prog}(\sigma_i, \phi_2) \vee (\mathsf{prog}(\sigma_i, \phi_1) \wedge \phi_1\,\mathsf{U}\,\phi_2)$

Table 1: Semantics of progression functions for different logical and temporal operators. The progression function takes in the current state and LTL formula that is updated after application of the function.

For an LTL expression ϕ and a state σ_i, we define the semantics of the progression func-

tion $\mathsf{prog}(\sigma_i, \phi)$ as in Table 1. At each point in time i, we can therefore update the LTL expression to reflect which parts of the original formula have so far been satisfied or unsatisfied. We can do this because if a sequence of truth assignments (i.e., a trajectory) satisfies an LTL formula at time i if the formula progressed through ϕ_i is then satisfied at time $i+1$.

Program Representation: Formally, we represent LTL expressions (i.e., executable programs) as a sequence of tokens that describe a possibly recursive sequence of functions in postfix notation. Every token in an expression is either a function of fixed arity (i.e., one or two arguments), constant, variable or λ term that is used to define Boolean functions. Previous work that parses to logical representations uses atomic types such as `Boolean`, `Integer`, `Size`, `Shape`, `Colour`, `Side` or composite types `Set(?)`, `Func(?, ?)`. For our purpose, since our LTL task specifications specify goal locations in the environment, our LTL vocabulary consists of observable elements in the environment of atomic types (`Object`, `Floor`, `Wall`) and either unary or binary logical operators (`Func`). All valid and syntactically correct programs have a return type of `Boolean`. For example, the LTL expression `( a ∧(⋄ b))` converted to postfix notation `( a b ⋄ ∧)` is linearised to allow easier execution with a stack based on the semantics of operators and propositions.

3.2 Planning in an Environment

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* respectively, $T : S \times A \times S \rightarrow [0, 1]$ is the *transition function*, $R : S \times A \times S \rightarrow \mathsf{Pr}(R)$ is the *reward function* and γ is the *discount factor*. An agent can take an action (from the set of all actions) and change its state, to move within states according to the transition function, hence accumulating reward according to the reward function. An agent learns a *policy* π i.e., a probability distribution over state and action pairs, that allows it to determine the actions it should take in each state with probability $\pi(a|s)$. The *state-action value* or *Q-value* denoted $Q^\pi(s, a)$ is the expected discounted return of selecting action *a* in state *s* and then selecting actions according to π, and a policy is *optimal* if the expected discounted reward gained by following the pilicy is maximal for every state $s \in S$. Given the Q-value function Q^*, the optimal policy is to then select the action *a* in every state *s* with the highest value of $Q^*(s, a)$.

World	Operation	Complexity
Database	execution	$O(\|W\| \times \|s\|)$
SQL: `SELECT*MAX(state.area) FROM state`		
CCG parse: $\lambda x.$ `flight`$(x) \wedge$`from`(x, bos)		
Gridworld	progression	$O(\|S\| \times \|s\|)$
	execution	$O(\|S\|^{\|s\|})$
LTL: `at_lamp U ⋄ (at_chair)`		
CCG: λ `a.move(a)` $\wedge$ `post(a,intersect` (λ`x.chair(x),you))`$\wedge$`pre(a,front(you,` λ`x.lamp(x)))`		

Table 2: Comparing complexities of different feedback systems for different world representations and logical forms. The first rows show example execution models in other domains i.e., SQL query or a CCG parse (lambda calculus expression) executed against a database, while the latter section compares feedback methods in navigation domains i.e., progression vs. execution or planning for LTL forms or a CCG parse.

Q-Learning: The Q-learning algorithm is a well-known off-policy algorithm, that learns a *target* policy while using some other *behaviour* policy for action selection. We begin the Q-learning process by initialising the Q-values of all state-action pairs to zero, and at every step

use some behaviour policy to pick an action *a* for the current state *s*, therefore returning a new state *s'* an reward *r* from the environment. The estimation of $Q(s, a)$ at the current timestep is then updated as shown in the equation below where α is the *learning rate*.

$$\text{Q(s, a)} \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

The Q-learning algorithm is guaranteed to converge to optimal Q-values if the behaviour policy visits every state-action pair an infinite number of times. In practice, this is done by setting the behaviour policy to be ϵ-greedy on the target policy, to ensure that on each step, the behaviour policy selects random actions with probability ϵ and the action with the highest Q-value with probability $1 - \epsilon$.

Function Approximation and DQN: For the Q-learning algorithm to keep track of state-action pairs, in the simplest form of Q-learning, a Q-table is used to store the required values. This however, is impractical for large (or infinite) state spaces. Therefore, we instead use some form of *function approximation* on the Q-value function i.e., we define the Q-value function as a function of state features, where Q-value updates involve updating the function, instead of updating entries in a Q-table. *Deep Q-Networks (DQN)* (Mnih et al., 2015) use neural networks for this function approximation of the Q-value function. An experience replay buffer and a target network are used to stabilise learning and the agent's experiences (of the form (s, a, r, s')) are stored in the bugger and sampled to train the network over time. The Q-learning updates are computed with respect to a *target* network which is only updated periodically to attempt to decrease the chance of the policy diverging.

Learning with Temporal non-Markovian Rewards: In order to allow handling of temporal constraints, we use the concept of a non-Markovian reward decision process that incorporates the LTL expression into the environment MDP. Instead of only considering the previous state, the reward function R is therefore defined over state histories $R : S* \rightarrow \mathsf{Pr}(R)$. Given this MDP, the Q-value function of a policy π is then defined over sequences of states:

$$\text{Q}^{\pi}(\langle s_0, .., s_t \rangle, a) = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R(\langle s_0, .., s_{t+k+1} \rangle) | A_t = a\right]$$

The reward functions for these MDPs are defined in terms of completing the task defined by the LTL formula. We incorporate the LTL task specification through a *labelling function* $L : S \rightarrow 2^P$ where P is the set of propositional symbols. This is therefore a 7-tuple $\tau = \langle S, A, T, P, L, \phi, \gamma \rangle$, where S, A, T and γ are defined as in an MDP and refer to states, actions, transitions and the discount factor respectively. P is the set of propositional symbols that form LTL task specifications, L is the labelling function defined above and ϕ is the set of tasks. As in general RL settings, the agent does not know the transition probability distribution of the domain, but has access to the labelling function and the task.

3.3 Environments and Data

SAIL from (MacMahon et al., 2006) is a navigation dataset containing route instructions annotated with trajectories for three different environments, each composed of connected hallways with different patterns (grass, brick, wood, gravel, blue, flower, or yellow octagons) on floors, paintings (butterfly, fish, or Eiffel Tower) on walls and objects (hat rack, lamp, chair, sofa, barstool, and easel) at intersections. Figure 1 shows an example environment annotated with object positions, wall paintings and floor patterns.

The challenge of learning to ground natural language to logic stems from the fact that instructions given by humans are complex, free-

No. of instruction sequences	706
No. of sentences	3233
Avg. sentences per sequence	4.61
Avg. tokens per sentence	7.94
Vocabulary size	522
Avg. trajectory length (per sentence)	3.05
Max. trajectory length (per sentence)	31

Table 3: Statistics from the SAIL dataset from (MacMahon et al., 2006) that contains natural language instructions annotated with trajectories.

form and of variable length (single sentences in isolation or full paragraphs). While this task and dataset bears superficial similarity to other navigation benchmarks, the language and policies required for this task are quite different — the proportion of instructions to actions is much higher, the interpretation of language is highly compositional and the length of the instructions vary widely. This task has therefore been the subject of focused attention in semantic parsing, resulting in a range of different approaches that attempt to learn how to plan in such settings.

3.4 Model

Our model draws insights from previous work (Guu et al., 2017; Goldman et al., 2017) that train semantic parsers from denotations with an algorithm that searches through a space of programs at training time, in order to find the correct program. We set this up as a reinforcement learning problem, where our agent samples a token from the vocabulary to generate a sequence of tokens that in our case, form an LTL expression. At the end of this sequential prediction of tokens, each LTL expression is given a binary reward of 1 or 0, by progressing the LTL expression along the trajectory in the environment.

Our training samples are of the form (x, t) where x is a natural language instruction and t is a trajectory in the environment, and our model generates program tokens $z_1, z_2..$ from left to right using a neural encoder-decoder model (Sutskever et al., 2014). We encode every utterance x with a bidirectional LSTM (Hochreiter and Schmidhuber, 1997) to create a contextualised representation h_i for every input token x_i. Our decoder is then a feed-forward network with attention (Bahdanau et al., 2014) over the output from the encoder, that takes as input, the last K tokens that were decoded. 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 as shown in the equations below.

Encoder The utterance u_m is encoded with a bidirectional LSTM:

$$h_i^F = LSTM(h_{i-1}^F, \phi_u(u_m, i))$$
$$h_i^B = LSTM(h_{i-1}^B, \phi_u(u_m, i))$$
$$h_i = [h_i^F; h_i^B]$$

Decoder Let $f(z_{1:t-1})$ refer to the execution history and e_m to the input embedding to the decoder. We can then compute an attention vector c_t as follows:

$$q_t = ReLU(W_q[e_m; f(z_{1:t-1})])$$
$$\alpha_i \propto \exp(q_t^T W_a h_i) i - 1, ..., |u_m|)$$
$$c_t = \sum_i \alpha_i h_i$$

After combining the above i.e., concatenating q_t with c_t, we then produce the program token by token. Formally, we produce a distribution over the set Z of program tokens, computed with a softmax, as shown in the equation below, where $\phi_z(z_t)$ is the embedding for a token z_t.

$$p_\theta(z_t|x, z_{1:t-1}) \propto \exp(\phi_{z_t}^T W_s[q_t; c_t])$$

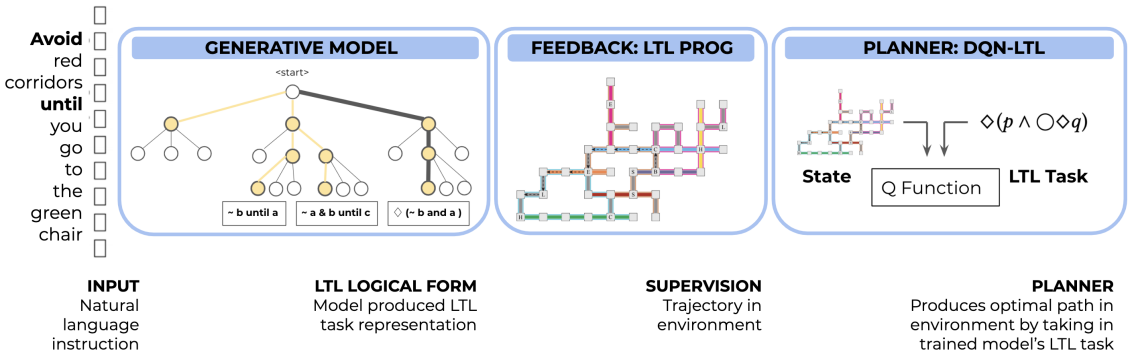

Figure 2: Skeleton of our weakly supervised model that learns to compose a logical meaning representation by only giving binary feedback signal through LTL progression against a ground truth trajectory. This model can then produce intermediate logical forms for any instruction that can then be given to a planner that solves the task to find a path.

We also keep track of the *execution history* i.e., the k most recent tokens $z_{t-k:t-1}$ and concatenate their embeddings. Our exploration strategy is the same as in (Guu et al., 2017) that uses an ϵ *-greedy randomised beam search* to deal with the problem of biased exploration that can result in visiting only a few states during exploration. Like regular beam search, at each iteration, the set of all continuations for the next token are sorted by the model probability $p_\theta(s|x)$. However, we inject random noise into exploration. Instead of simply selecting the k highest scoring continuations (for a beam of size k), we choose continuations one by one without replacement, from the set of all possible continuations. At each selection step from the remaining pool, we either unformly sample a random continuation with probability ϵ or pick the highest scoring continuation from the pool with probability $1 - \epsilon$.

Figure 2 shows a skeleton of our proposed model that is trained to compose together an LTL formula for the input natural language utterance, supervised by a trajectory, by progressing the LTL expression along this trajectory in the environment. This gives us a score of the satisfiability of the produced LTL expressions that allows ranking and learning of logical meaning representations.

For the weakly supervised model, we perform a grid search over hyperparameters to maximise accuracy over the validation set. These hyperparameters include the learning rate, the value of ϵ and the embedding size.

Planning Mechanism: To plan in a given environment with the grounded LTL task specification, we use Deep Q-Networks with LTL specifications, following the methodology from (Littman et al., 2017) to learn LTL-based rewards. We use standard RL to solve the cross-product MDP where the input to the Q-value function is both the state and the progressed LTL task at every step. This is the state-of-the-art approach to learn with reward functions specified in LTL to allow us to deal with temporal non-Markovian constraints.

The DQN implementation is baed on the OpenAIBaselines (Dhariwal et al., 2017). To train the network, we use *experience replay* — minibatches of experiences are sampled from an experience replay buffer of a fixed size. The feature vector at each state is computed based on the distance of every object from the agent. We use feedforward networks with 2 hidden layers and 64 ReLu units, trained us-

ing an Adam optimiser with a learning rate of 0.0001. At every step, the DQN network learns by randomly sampling 32 transitions from the replay buffer. We set the size of the buffer to 25,000 and update the target network at every 100^{th} step and use a discount factor of 0.9.

4 Experimental Evaluation

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 instructions in the test set. Models are tested on both the single and multi-sentence instructions to assess their ability to correctly interpret the intent of the instruction and navigate to the goal state in the environment.

Comparison to prior work: We compare our model to existing work that report results on the SAIL dataset, by cross-validating over the three maps and reporting a final accuracy. Most similar to our setup is the approach from (Artzi and Zettlemoyer, 2013) that train a CCG semantic parser with supervision from trajectories, however they do this by manually building execution models for the feedback mechanism which is more computationally intractable — the worst case complexity is exponential in the number of variables.

System	Single	Multi
Chen and Mooney (2011)	54.4	16.18
Chen (2012)	57.28	19.18
+ additional data	57.62	20.64
Kim and Mooney (2012)	57.22	20.17
Artzi and Zettlemoyer (2013)	65.28	31.93
Andreas and Klein (2015)	59.60	-
Mei et. al. (2017)	71.05	30.34
Ours	66.92	20.17

Table 4: Evaluation of systems on the SAIL dataset. The second and third columns shows accuracies of the reaching the goal state for commands that take the form of single sentences or entire paragraphs.

Goal-state accuracy To directly compare our approach with existing work, measure the *goal-state* accuracy i.e., the ability of the model to reach the correct final location in the environment after correctly interpreting and grounding the natural language instruction. Specifically, for each instruction in the test set, our models produces an LTL logical form, given to the DQN-LTL model that performs Q-learning to solve the navigation task to reach the final goal state. As shown in table 4 we compare 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., final goal-state accuracy over the SAIL dataset. These include algorithms supervised with logical forms (Chen and Mooney, 2011; Chen et al., 2015) that learn semantic parsers for natural language instructions as well as ones that involve strategies for online learning of lexicons for the semantic parsers (Chen, 2012) and ones that use contextual information (Chen et al., 2010) for better language understanding. We also compare models supervised with paths — the supervised alignment-based models (Andreas and Klein, 2015) that build grounding graphs representations to execute instructions and neural sequence-to-sequence models (Mei et al., 2016) that translate natural language instructions to actions that an agent can execute in the environment.

Path accuracy 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, in complicated environments with several possible paths. Figure

5 Conclusion

We propose a weakly supervised semantic parsing model that requires no supervision

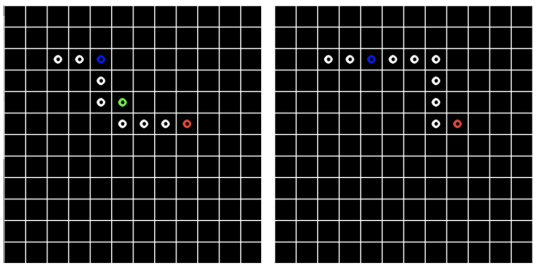

Figure 3: Example possible paths that require more fine-grained evaluation than just final goal location. While both paths in the figure reach the correct goal location (and get perfect accuracy under that metric) only the one on the left follows the constraints specified by the instruction.

System	Prec.	Recl.	Acc.
Mei et. al. (2017)	31.2	30.19	91.2
Ours	33.5	35.2	94.3

Table 5: Evaluating paths produced by systems against ground-truth paths. Each path is represented as a masked vector of locations in the environment, thus allowing comparing of precision, recall and accuracy against the gold path.

of ground-truth logical forms during training. Moreover, we propose to ground natural language to a formal language that allows handling of complex, temporal events that are typically unable to be handled by most (traditionally Markovian) methods. Our framework exploits the structure of LTL by progressing grounded LTL expressions along paths in the environment, thus giving the required supervision signal to parse language into logical form. As opposed to other methods that employ sequence-to-sequence models to map language directly to actions in the environment, our method first formulates the meaning of the natural language instruction in logical form that is interpretable, and then uses this meaning representation to formulate a plan to navigate in an environment.

6 Future Work

Our concern in this project is to ground natural language instructions to logical representations that preserve the underlying meaning of the instruction that agents can then later use to correctly navigate in an environment to solve the task. Here, we primarily focus on the semantic parsing component, and show that grounding to LTL

While this work does not plan or execute a policy in an end-to-end fashion, the use of LTL for task specifications can be beneficial for learning to plan, especially in hierarchical reinforcement learning settings that attempt to decompose tasks into subtasks. Future work will explore learning to ground language to collections of tasks specified in LTL, that can be composed together and learned in RL settings, by allowing agents to make use of LTL progression to extract shared subtasks and generalise this across different instances.

References

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

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

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

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

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on free-

base from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544.

Satchuthananthavale RK Branavan, Harr Chen, Luke S Zettlemoyer, and Regina Barzilay. 2009. 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.

David L Chen. 2012. 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.

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

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

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

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. 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.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. 2017. Openai baselines. *GitHub, GitHub repository*.

Juraj Dzifcak, Matthias Scheutz, Chitta Baral, and Paul Schermerhorn. 2009. 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 *2009 IEEE International Conference on Robotics and Automation*, pages 4163–4168. IEEE.

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

Omer Goldman, Veronica Latcinnik, Udi Naveh, Amir Globerson, and Jonathan Berant. 2017. Weakly-supervised semantic parsing with abstract examples. *CoRR*, abs/1711.05240.

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

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

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

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

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

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

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

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

Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. 2006. Walk the talk: Connecting lan-

guage, knowledge, and action in route instructions. *Def*, 2(6):4.

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

Dipendra Misra and Yoav Artzi. 2015. Reinforcement learning for mapping instructions to actions with reward learning.

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

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

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

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

Lappoon R Tang and Raymond J Mooney. 2000. 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.

Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. 2018. 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.

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

Edward C Williams, Nakul Gopalan, Mine Rhee, and Stefanie Tellex. 2018. 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.

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

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