Learning to Ground Language to Temporal Logical Form

Roma Patel [*]
Brown University

Stefanie Tellex
Brown University

Ellie Pavlick
Brown University

Abstract

Natural language commands often exhibit sequential constraints, e.g. *"go through the kitchen and then into the living room"* that (traditionally Markovian) methods in reinforcement learning (RL) cannot adequately handle. To express these constraints, we propose to ground language to Linear Temporal Logic (LTL) but propose to do this when direct supervision is not available, i.e. 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 logical forms that satisfy ground-truth paths. We then use the LTL expressions to plan in the environment using off-policy RL. We evaluate our framework on both goal state and path accuracy and demonstrate competitive performance on goal state accuracy and higher performance on path accuracy. We analyse the instructions and see that the largest gains come from the ability of our model to handle temporal instructions.

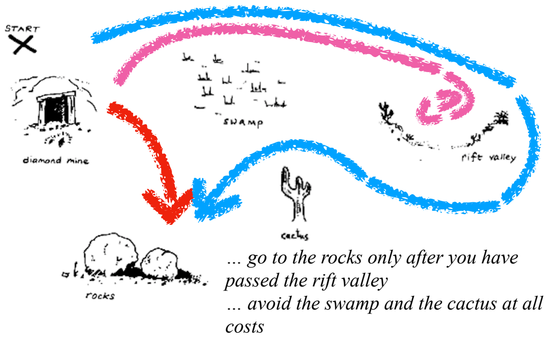

START
diamond mine
SWAMP
rift valley
cactus
rocks
... go to the rocks only after you have passed the rift valley
... avoid the swamp and the cactus at all costs

Figure 1: Navigational instructions often require keeping track of events that happen before the previous state, thus breaking Markovian assumptions. While both red and blue (and many possible) paths in the figure reach the goal, only the blue correctly follows the specified temporal constraints.

1 Introduction

A typical instruction following task is challenging for two reasons. First, the intended meaning of the natural language instruction must be correctly grounded in order to solve the task. Second, at task-solving time, agents have to learn policies that not only reach the final goal location, but also satisfy the specified constraints. The complex (e.g., temporal, sequential, conditional) constraints are nontrivial to express and achieve by conventional methods in classical reinforcement learning (RL) that assume a Markovian reward function, thus relying on only the previous state. For example, commands that specify temporal ordering such as *"go to A only after you have gone to B"* could potentially map to unbounded action sequences, as shown in Figure 1, unless we have adequate representation of state properties in accordance with temporal patterns over time. These temporal constraints occur frequently in natural language, but are an obstacle to (traditionally Markovian) planning approaches. To allow representation of such constraints, we propose to learn a semantic parser that grounds natural language instructions to Linear Temporal Logic (LTL) meaning representations. We furthermore do this with weak supervision by learning from demonstrations: while collecting and annotating symbolic logical forms is an expensive process, we can more easily collect *executions* of these as trajectories in an environment. Unlike other symbolic forms, LTL not only allow us to express temporal constraints but allows the use of more efficient feedback methods like LTL progression, to check satisfiability of logical forms against ground-truth paths. We use the grounded LTL representations to plan in an environment using Q-learning with LTL-based rewards. We evaluate our method on a benchmark dataset in terms of both path and end-state accuracy. We show competitive performance on goal-state accuracy while improving fidelity to the temporal constraints, as measured by path accuracy. Moreover, our weakly-supervised training method is substantially more efficient, providing feedback via a linear-time computation as opposed to the exponential-time execution models used in prior work.

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, unlike previous work, we propose a more fine-grained evaluation of the *path* taken, rather than only the goal location. 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.

2 Related Work

In the semantic parsing literature, previous work has explored weak supervision methods (Artzi and Zettlemoyer, 2013; Krishnamurthy and Mitchell, 2012) that execute produced logical forms to determine final goal locations. All such approaches ground to lambda calculus expressions and logical forms other than LTL that cannot handle temporal order. In the RL literature, previous work has used LTL to formulate tasks, 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 tasks 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 natural language to temporal logic without supervision of logical forms. Below, we enumerate the different avenues explored by previous instruction following methods.

Language $\rightarrow$ Logical Form There is ample prior work on supervised semantic parsing to logical forms other than LTL (Zettlemoyer and Collins, 2012; Tang and Mooney, 2000; Berant et al., 2013; Yu et al., 2018) and to LTL (Gopalan et al., 2018). More relevant to our work is previous work that 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 (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). Most relevant to our work is the model of (Artzi and Zettlemoyer, 2013), however our work is different in that it handles temporal order and also provides a more efficient feedback mechanism.

Language $\rightarrow$ Plan Also relevant is the body of work which seeks to map natural language directly to action sequences, e.g. Mei et al. (2016); 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. 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 (Koehn and Knowles, 2017; Lake and Baroni, 2017).

Logical Form (LTL) $\rightarrow$ Plan Previous work (Dzifcak et al., 2009; Gopalan et al., 2018) has explored grounding language to LTL and then planning with 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 generalise to unseen logical forms. Recent approaches have used Deep Q-Networks with LTL specifications (Toro Icarte et al., 2018), by making use of LTL based rewards (Littman et al., 2017) where the input to the Q-value function is both the state and 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. While this line of work highlights the benefits of LTL for temporal abstraction, it does not explore methods to first ground variable language to LTL for futher planning.

3 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*. 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., $\diamond\phi = \mathsf{true}\,\mathsf{U}\,\phi$.

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, thus allowing us to check existence of propositions in states for progression functions in Table 1.

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. Table 1 shows how we define our progression functions.

4 Model

Our model draws insights from previous work (Guu et al., 2017; Goldman et al., 2017) that train semantic parsers from denotations, by searching through a space of programs at during training. 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.

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` $\wedge$ `(` $\diamond$ `b ) )` converted to postfix notation `a b` $\diamond$ $\wedge$ can be realised in an environment where the propositions correspond to locations. 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 (e.g., *stool, brick corridor..* etc).

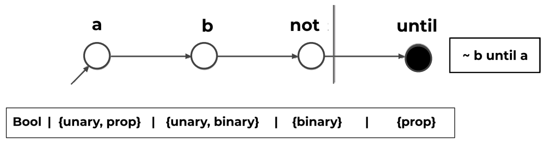


Figure 2: Using a stack for type-constrained decoding to ensure syntax of produced logical forms.

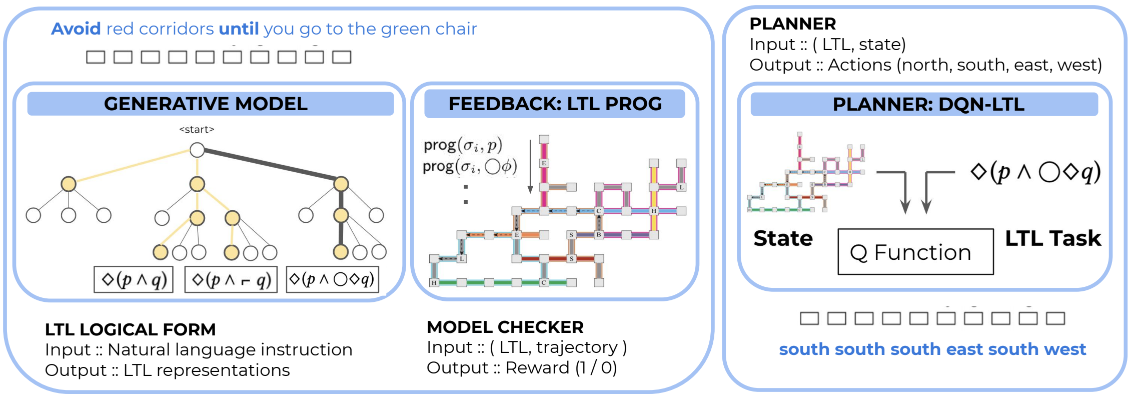
Encoder and Decoder: Our training samples are of the form (x, t) where x is a natural language instruction and t is a trajectory in the environment. As in (Guu et al., 2017) our model generates program tokens z_1, z_2.. from left to right using a neural encoder-decoder model (Sutskever et al., 2014). 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 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.

prog(σ_i, p)	true if $p \in \sigma_i$, where $p \in P$
prog(σ_i, p)	false if $p \notin \sigma_i$, where $p \in P$
prog$(\sigma_i, \neg\phi)$	$\neg$prog(σ_i, ϕ)
prog$(\sigma_i, \phi_1 \wedge \phi_2)$	prog$(\sigma_i, \phi_1) \wedge$ prog(σ_i, ϕ_2)
prog$(\sigma_i, \phi_1 \vee \phi_2)$	prog$(\sigma_i, \phi_1) \vee$ prog(σ_i, ϕ_2)
prog$(\sigma_i, \bigcirc\phi)$	ϕ
prog$(\sigma_i, \phi_1 \, \mathsf{U} \, \phi_2)$	prog$(\sigma_i, \phi_2) \vee ($prog$(\sigma_i, \phi_1)$ $\wedge \; \phi_1 \, \mathsf{U} \, \phi_2)$

Table 1: Semantics of progression functions that take in the current state σ_i and LTL formula ϕ that is updated after application of the function.

Supervision: Each logical form gets a binary (1 or 0) reward by *progressing* it along the ground-truth path, using the progression functions defined in Table 1. 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 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[1]

[1]We note however, that in the SAIL environment there are very few instances of such utterances occurring.



Avoid red corridors until you go to the green chair
GENERATIVE MODEL
<start>
$\Diamond(p \wedge q)$
$\Diamond(p \wedge \neg q)$
$\Diamond(p \wedge \bigcirc\Diamond q)$
FEEDBACK: LTL PROG
prog(σ_i, p)
prog$(\sigma_i, \bigcirc\phi)$
LTL LOGICAL FORM
Input :: Natural language instruction
Output :: LTL representations
MODEL CHECKER
Input :: (LTL, trajectory)
Output :: Reward (1 / 0)
PLANNER
Input :: (LTL, state)
Output :: Actions (north, south, east, west)
PLANNER: DQN-LTL
$\Diamond(p \wedge \bigcirc\Diamond q)$
State
LTL Task
Q Function
south south south east south west

Figure 3: Figure shows the pipeline of our weakly supervised model. The generative model on the left learns to compose a logical form with binary feedback. Once trained, the planning component on the right takes in LTL logical forms to produce actions in the environment.

4.1 Planning in an Environment

Once we produce LTL interpretations of instructions, we use Q-learning with LTL task specifications to formulate a plan. We briefly go over the planning mechanism, that takes into account the state and LTL task, to select actions to navigate to the goal location.

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*, $T : S \times A \times S \rightarrow [0, 1]$ is the *transition function*, $R : S \times A \times S \rightarrow \Pr(R)$ 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 $\pi(a|s)$. The *state-action value* or *Q-value* denoted $Q^{\pi}(s, a)$ allows attribution of values to states and actions to enable action selection at every state.

Q-Learning and DQN: The Q-learning algorithm is an off-policy algorithm that aims to learn Q-values for states. It learns a *target* policy while using some other *behaviour* policy for action selection. At every step, the Q-learning process uses a behaviour policy to pick an action *a* for a state *s* to return a new state *s'* and reward *r* from the environment. The estimation of $Q(s, a)$ at the current timestep is then updated as in the equation below where α is the *learning rate*.

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

In the simplest form of Q-learning, a Q-table is used to store required values for states, which is impractical for large (or infinite) state spaces. An alternative is to use some form of *function approximation* on the Q-value function. We use *Deep Q-Networks (DQN)* (Mnih et al., 2015) to employ neural networks to approximate the Q-value function with an experience replay buffer and target network that stabilise learning. 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.

DQN with LTL: To handle temporal constraints, we use a non-Markovian reward decision process that incorporates the LTL expression into the environment MDP. The reward function R is therefore defined over state histories $R : S* \rightarrow \Pr(R)$ instead of only the previous state. The Q-value function of a policy π

is then defined over sequences of states:

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

We incorporate the LTL task with a *labelling function* $L : S \rightarrow 2^P$ from states to propositions P. This MDP 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 ordinary MDP, P is the set of propositional symbols, L is the labelling function defined above and ϕ is the set of tasks. As in general RL, the agent does not know the transition probability distribution of the domain, but has access to the labelling function and the task. 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. The DQN implementation is based 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 feed-forward networks with 2 hidden layers and 64 ReLu units, trained using 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. Once trained, this function approximation allows us to get optimal actions at every state for a given LTL task. This entire path taken (i.e., the sequence of actions) for each instruction is evaluated as explained in the next section.

5 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.

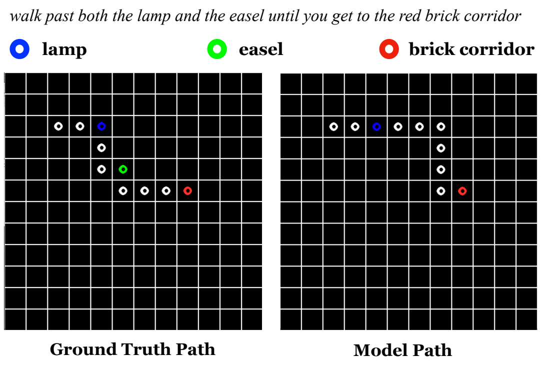
5.1 Environments and Data

SAIL from MacMahon et al. (2006) 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. 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 (Anderson et al., 1991; Bugmann et al., 2001) , 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.

5.2 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


walk past both the lamp and the easel until you get to the red brick corridor
lamp
easel
brick corridor
Ground Truth Path
Model Path

METRICS		Edit distance:	5
Goal state accuracy:	100%	Recall:	44.44%
Precision:	36.36%	Accuracy:	91.66%

Figure 4: Example paths that require more fine-grained evaluation than just final goal location. While the path on the right gets perfect accuracy under a metric evaluating final locations, it is incorrect in that it doesn't follow specified constraints.

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. Figure 4 shows two example paths in a simple grid-world that both reach the goal location. 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.

6 Results

Comparison to prior work: 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 Artzi and Zettlemoyer (2013) that trains a CCG semantic parser supervised by trajectories. Other methods include algorithms supervised with logical forms (Chen and Mooney, 2011; Chen et al., 2015) that learn semantic parsers for instructions as well as ones that involve strategies for online learning of lexicons (Chen, 2012) and ones that use contextual information (Chen et al., 2010) for better language understanding. We also compare to the supervised alignment-based models (Andreas and Klein, 2015) that build grounding graph representations to execute instructions and neural sequence-to-sequence models (Mei et al., 2016) that translate language to actions in the environment.

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. (2016)	71.05	30.34
Ours	66.92	20.17

Table 2: 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.

Performance: Table 2 shows goal-state accuracy of systems on SAIL, while Table 3 shows evaluation metrics for paths produced, compared to ground-truth paths in the dataset. Table 2 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 3 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.

Data	System	Prec.	Recl.	Acc.	ED
All	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	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

Table 3: For path evaluation, we evaluate precision, recall, accuracy (higher is better) and edit distance (lower is better), compared to the previous best model; *Seq2seq* refers to (Mei et al., 2016) while *Ours* is our weakly supervised method.

Complexity: An important contribution of our approach is the efficiency of our feedback mechanism over previous execution models. Previous work built execution models for CCG parses of instructions, while our work makes use of LTL progression to reward logical forms. Table 4 shows the difference in complexities for either *progressing* an expression along a sequence of truth assignments as opposed to *executing* it by building models for semantic types in the expression.

World	Operation	Complexity
Gridworld	progression	$O(\|S\| \times \|s\|)$
	execution	$O(\|S\|^{\|s\|})$
`LTL: at_lamp U` $\diamond$ `(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 4: Comparing complexities of different feedback systems in navigation domains i.e., progression vs. execution for LTL forms or a CCG parse.

7 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 propose a more fine-grained evaluation of the path taken, rather than just the goal location, and show that our approach more closely adheres to path constraints than all previous models, while also showing comparable performace on reaching the goal location. 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. Future work involves incorporating the structure of LTL to jointly learn logical forms and execution models, especially in domains that specifically require temporal reasoning.

References

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

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 freebase 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.

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

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.

Joohyun Kim and Raymond J Mooney. 2012. 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.

Philipp Koehn and Rebecca Knowles. 2017. Six challenges for neural machine translation. *arXiv preprint arXiv:1706.03872*.

Jayant Krishnamurthy and Tom M Mitchell. 2012. 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.

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.

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

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 language, 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, 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*.