Learning to Parse Natural Language to Grounded Reward Functions with Weak Supervision

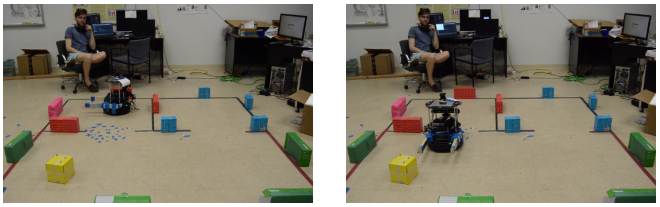
Edward C. Williams* and Nakul Gopalan* and Mina Rhee and Stefanie Tellex

***Abstract*— In order to intuitively and efficiently collaborate with humans, robots must learn to complete tasks specified using natural language. We represent natural language instructions as goal-state reward functions specified using lambda calculus. Using reward functions as language representations allows robots to plan efficiently in stochastic environments. To map sentences to such reward functions, we learn a weighted linear Combinatory Categorial Grammar (CCG) semantic parser. The parser, including both parameters and the CCG lexicon, is learned from a validation procedure that does not require execution of a planner, annotating reward functions, or labeling parse trees, unlike prior approaches. To learn a CCG lexicon and parse weights, we use coarse lexical generation and validation-driven perceptron weight updates using the approach of Artzi and Zettlemoyer [4]. We present results on the Cleanup World domain [18] to demonstrate the potential of our approach. We report an F1 score of 0.82 on a collected corpus of 23 tasks containing combinations of nested referential expressions, comparators and object properties with 2037 corresponding sentences. Our goal-condition learning approach enables an improvement of orders of magnitude in computation time over a baseline that performs planning during learning, while achieving comparable results. Further, we conduct an experiment with just 6 labeled demonstrations to show the ease of teaching a robot behaviors using our method. We show that parsing models learned from small data sets can generalize to commands not seen during training.**

I. Introduction

Natural language provides an expressive and accessible method for specifying goals to robotic agents. These commands constitute a significant subset of useful behaviors that a user can specify in collaborative scenarios. For example, a user of a household robot can easily specify navigation (e.g., "go to the TV room") or manipulation (e.g., "place the dishes in the sink") goals in natural language without needing to learn complex APIs or user interfaces. Such commands describe a desired configuration of the world, and expect a robotic agent to modify the environment accordingly. These commands are easily modeled as goal conditions to be satisfied by a planning agent.

In this paper, we represent these goal conditions as lambda-calculus goal-based reward functions under the Markov Decision Process (MDP) [7] formalism, which give the agent positive reward for reaching the goal state. Planning then aims to maximize the sum of rewards the agent receives while attempting to reach the goal condition optimally. Planning in the MDP formalism is useful in the setting of robotics as it allows for a natural method to deal with stochasticity. The use of reward functions as an intermediate representation allows a separation between the language interpretation and planning components.



(a) State at task initiation (b) State at task completion

Fig. 1: The figure shows an example pair of states of the Turtlebot before and after it performed the command to "go to the largest room." The robot moves from the red room to the large green room, marked using the colored blocks. The robot plans the trajectory using a reward function generated from the natural language command on the fly using a CCG parser we learned using weak supervision from language.

We learn a mapping from natural language into the space of lambda-calculus expressions using a semantic parser. This mapping is modeled as a weighted linear Combinatory Categorial Grammar (CCG) [9, 24] parser, which pairs words in a lexicon with lambda-calculus representations of their meaning. More importantly, a CCG provides a mechanism for composing lambda-calculus expressions representing sentences from subexpressions representing their constituent words and phrases. Our lambda-calculus representation interfaces with Object Oriented MDP (OO-MDP) [10] states while enabling the use of complex commands involving comparators, object attributes, relationships between objects, and nested referring expressions.

Existing approaches for learning to map sentences to reward functions require fully supervised data, including complete reward function specifications paired with natural language commands [18, 6]. This requires manual annotation of natural language datasets, and the resulting learned models have a limited ability to generalize to unseen tasks. As well, existing CCG-based approaches that map natural language to trajectory specifications [4, 5] require executing planning during learning to validate proposed logical expressions, which is computationally expensive and does not allow planning in Markov Decision Processes. Our method allows for efficient learning using only pre- and post-condition states as annotation, instead of complete reward functions or their derivations, which are used to validate reward functions

Department of Computer Science, Brown University, Providence, RI 02912, USA {edward_c_williams@, ngopalan@cs., mina_rhee@, stefie10@cs.}brown.edu
* The first two authors contributed equally.

without executing a planner.

We believe that ours is the first approach to produce compositional reward functions using a CCG semantic parser, and then use the reward function for the purposes of planning on an agent in an MDP. We trained and evaluated our model on data collected from Amazon Mechanical Turk (Figure 2) on simulated mobile manipulation tasks in Cleanup Domain [18]. Our results demonstrate effective learning of a weakly supervised parser with an F1 score of 0.82 on a corpus of 23 behaviors. We compare against baselines that use planning to validate parses during learning, and show that our method achieves comparable performance with orders of magnitude improvement in computation time during learning. Moreover, we present a robot demonstration of our method in a Turtlebot mobile-manipulation task using 6 weakly annotated demonstrations to learn two behaviors, and also show generalization to unseen tasks.

II. Related Work

Zettlemoyer and Collins [27] presented a supervised method to learn CCG parsers given natural language annotated with lambda-calculus logical forms. Krishnamurthy and Mitchell [15] demonstrated a weakly supervised method for learning CCG semantic parsers given a syntactically parsed sentence and a knowledge base. Artzi and Zettlemoyer [4] extended this idea of weak supervision to planning in a navigation task by mapping natural language to a logical form specifying a trajectory. The trajectories produced by the CCG parser while learning were compared to demonstration trajectories for a validation signal. However, these plans were not generated on an MDP, and the semantic representations of language directly represented sequences of actions rather than goal conditions. Planning on a physical robot requires modeling stochasticity, which is modeled well using MDPs, to allow agents to recover from failure. As we are only interested in goal based commands, it is not necessary to validate using trajectory information during learning. Instead, we can learn from pre- and post- condition states provided as demonstrations for a natural language command.

Tellex et al. [25] described a method using syntactic parse trees of language to create probabilistic graphical models (PGMs) that can create trajectories that match the natural language phrase. This process of generating trajectories is expensive, hence Howard et al. [13] used the PGMs to generate constraints that be used to plan, separating the language interpretation and planning components of the system. Our method also produces constraints on the end state of a robot's trajectory. While Howard et al. [13] assumes access to a pre-trained syntactic, our method needs pre-defined logical expressions to test attributes of objects. A related line of research (MacGlashan et al. [18], Arumugam et al. [6]) looks to translate natural language to sets of reward functions so that agents can perform planning within an MDP. These methods model the process of mapping natural language to reward functions as sequence to sequence translations [18] and multi-class classification [6] using a relatively simple semantic representation and without the ability to learn and interpret nested referential expressions. We instead represent the task as semantic parsing with a highly compositional semantic representation. Portions of this paper appeared in Williams et al. [26] describing an early attempt to parse natural language to reward functions with weak supervision.

Inverse Reinforcement Learning (IRL) [20] is a method for learning a reward function in an MDP using demonstration trajectories and a feature representation of the state space. In our work, we learn a compositional reward function from pre-specified predicates, natural language, and pre- and post- condition states provided as supervision. IRL learns a richer class of reward functions than the goal-based reward functions we learn in this work. However, our use of natural language to learn compositional reward functions allows for generalizability of the learned reward functions as described in Section VII. Another series of approaches map natural language directly to policies in an MDP, learned from demonstrations provided at training time [19, 14]. However, these methods have not been demonstrated to work with small training corpora, which is enabled in our case by the seed lexicon provided to our semantic parser.

III. Task Domain

Markov Decision Processes (MDPs) [7] model stochasticity, which is an important factor when trying to plan or learn with robots in the physical world. Hence, we represent our robot manipulation and navigation domain as an MDP. An MDP is described by a five-tuple ($\mathcal{S}$,$\mathcal{A}$, $\mathcal{T}$, $\mathcal{R}$, $\mathcal{E}$). Here $\mathcal{S}$ represents the environment's state space; $\mathcal{A}$ represents the agent's action space; $\mathcal{T}(s, a, s')$ is a probability distribution defining the transition dynamics (i.e., the probability that a transition to state s' occurs when action a is taken in state s); $\mathcal{R}(s, a, s')$ represents the reward function returning the numerical reward that the agent receives for transitioning to state s' with action a from state s; and $\mathcal{E} \subset \mathcal{S}$ is a set of terminal states that, once reached, prevent any future action. The goal of planning in an MDP is to find a *policy*—a mapping from states to actions—that maximizes the expected future discounted reward. In such a domain, we can define goal-state reward functions as functions that, for a given set of terminal goal states $G \subset \mathcal{E}$, produce the following output:

$$R_G(s, a) = \begin{cases} 1, & \text{if } s \in G \\ 0, & \text{otherwise} \end{cases}$$

We define our set of tasks as those that can be executed by planning over goal-state reward functions in the Cleanup World object manipulation domain [18]. We specify this MDP using Object Oriented MDPs (OO-MDP) [10], which provides a factored representation of an MDP problem. The factorization of the environment and actions is done using objects present in the environment, which is convenient as humans naturally describe world states with respect to objects present in it. Cleanup Domain [18] is an MDP environment generator with an agent, and different numbers of rooms and objects. The object have types, such as "basket", "chair", and "block" and attributes, such as "color"

and "location". The agent possesses location and orientation attributes. Rooms have attributes of "color", "location" and "size". In this work we use a three room configuration of Cleanup world with changing attributes for rooms and objects and the agent. The agent can take the actions of *north, south, east* and *west*. We have previously shown the stochastic nature of this domain, with doors that can lock and unlock and objects that are lost by the agent [11].

IV. Method

To map natural language to grounded reward functions, we define a compositional semantic representation that defines sets of goal states from relationships between objects in our OO-MDP domain. We then induce a weighted linear CCG parser [9] mapping natural language into our semantic representation, using pre- and post- condition states as supervision, rather than fully annotating language with semantics. Our learning algorithm is derived from the validation-based perceptron learner of Artzi and Zettlemoyer [4].

A. Semantic Representation and Execution

To bridge the gap between natural language and OO-MDP reward functions, we define a lambda-calculus semantic representation for language that defines a set of goal states in the OO-MDP domain. Lambda calculus, a computational formalism based on function application and variable binding, is well-explored as a semantic representation for natural language [8, 27, 28, 4]. Its utility for natural language representation comes from its expression of the compositionality of natural language while providing an interface between language and computation.

We make the assumption that natural language commands to our agent define a configuration of the world that the user would like the agent to produce, by re-arranging objects in the world or moving to a different location, is reflected in the nature of our representation - natural language is represented as a proposition function over states in the MDP. For instance, a complete lambda-calculus expression used as a reward function for the task in Figure 2, "go to the chair", is $near(the(\lambda x.agent(x)), the(\lambda y.red(y) \wedge chair(y)))$, which acts as as the goal-state reward function in Section III. This function is composed of relational operators ("*in*"), definite determiners ("*the*"), and noun phrases describing properties of objects. We adopt much of the notation from [4, 3, 27, 24] for our lambda-calculus functions. However, we eschew the neo-Davidsonian event semantics used by Artzi and Zettlemoyer [4], as our tasks are purely represented by state configurations. As such, we deliberately avoid modeling components of trajectories in our semantic representations, which preserves our efficient separation of language and planning components of the system, at a cost to the expressiveness of our semantic representation. Rather than implementing our CCG semantic representations as database queries [27] or trajectory specifications in a custom-built navigation domain [4], we implement our lambda-calculus functions as operating on objects in OO-MDP states.

We primarily model five components of natural language necessary to completing our tasks:

1) Nouns: We model nouns as single-argument lambda-calculus functions that map OO-MDP objects in a given state to Boolean values. For example, the phrase "block" would be represented as a function $\lambda x.block(x)$. When this phrase is evaluated in an OO-MDP state on a particular object o, it returns true if o is a block.

2) Adjectives: Adjectival language, such as "green block," are modeled as conjunctions of these single-argument proposition functions. The function $\lambda x.green(x) \wedge block(x)$ checks two attributes of the object provided as its argument. As an OO-MDP object is parameterized using object classes and attributes, these single-argument proposition functions can be implemented as lookup operations on object instances.

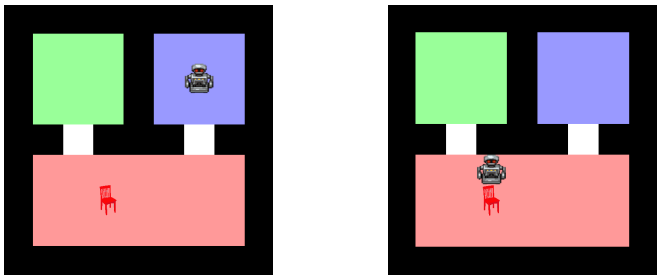
3) Definite Determiners: We model the definite determiner "the" as a function that maps a proposition function to an object that satisfies the given proposition function, following Artzi and Zettlemoyer [4]. This can be represented as a search over a set of objects in an OO-MDP state. We note that definite determiners are evaluated with respect to the initial state of the task, as we assume object references should be resolved at the world-state in which the natural language command was issued.

4) Comparators: Comparators, such as "the biggest" or "the smallest," are modeled as $argmax$ and $argmin$ operators over the entire space of objects with respect to a proposition function over objects and a numerical property of objects. The operator searches over all objects that satisfy the provided proposition function with respect to the comparison being made, and chooses the object that maximizes the numerical property. In our model, we only provided the argmax with a $size$ operator, although in principle the operator could be any single-argument function that returns a number from an object. Comparators, like definite determiners, are also resolved with respect to the initial world-state.

5) Relations: Spatial relationships between two objects are modeled as lambda-calculus functions of arity two. The function $\lambda x.\lambda y.in(x, y)$, for example, uses the spatial dimensions of the object provided as the second argument to determine if it contains the first argument. All relationship proposition functions produce Boolean outputs when evaluated in a given OO-MDP state. Currently, we model four spatial relationships: containment (in), adjacency (near), and directional adjacency to the right or left of the referent. All lambda-calculus functions were implemented as pre-specified JScheme [1] predicates that operate on states and objects in the BURLAP [17] reinforcement learning library.

B. Parser Learning

To map natural language to elements of our semantic representation, we learn a weighted linear CCG parser [9] that maps natural language commands $x \in X$ to a logical form $y \in Y$ in our semantic representation. We collect a data set where each element is of the form (x_i, S_i), where $x_i \in X$ is a natural language command, and S_i is a set of pairs of MDP states, as described in Section V. To learn this parser



(a) State at task initiation (b) State at task completion

Fig. 2: The figure shows an example pair used to collect data. Here we ask the users to give a command to the robot that will result in the pre- and post condition behavior shown in the pair of images.

without providing semantic representations as annotations directly, we define a validation function that determines if a given semantic parse will produce the correct behavior as described by our training data. We additionally use a modified form of the coarse lexical generation procedure of Artzi and Zettlemoyer [4] to produce new CCG lexical entries from training data. To learn parser weights, we use the validation-driven perceptron learning algorithm of Artzi and Zettlemoyer [4] as implemented in the Cornell Semantic Parsing Framework (SPF) [2].

1) Parser: Our learning objective is to learn a set of parser weights $\theta \in R^d$ for a weighted linear CCG parser [9] with a d-dimensional feature representation $\Phi(x, y)$. This parser uses a variant of the dynamic-programming CKY algorithm to produce the highest scoring parse $\hat{y}$ from a natural language command x. To perform training and inference, we use the linear CCG parser implemented in the Cornell SPF [2]. We added additional features to our model to induce correct behavior in situations where similar language describes different tasks depending on context. For example, the phrase "go to" either implies a containment or adjacency relationship depending on the object being referred to. Earlier iterations of the parser commonly confused our *in* and *near* predicates. To resolve this, we added features recording pairwise appearances of specific predicates.

2) Parse Validation: To facilitate our validation-driven perceptron learning, we define a parse validation function similar to those of Artzi and Zettlemoyer [4]. We define the function $\mathcal{V}(y, S) \in \{0, 1\}$, which takes as input a semantic parse y and set of pre- and post- condition state pairs $S = (S_i, S_f)$.

$$\mathcal{V}(y, S) = \begin{cases} 1, & y(S_i) = 0 \wedge y(S_f) = 1 \\ 0, & \text{otherwise} \end{cases}$$

We validate over all state pairs provided with a given natural language command. This function ensures that the semantic parse y correctly defines the desired set of goal states. As our tasks define sets of goal states, this is sufficient to check if a parse is valid without invoking a planner. For comparison, we tested against a baseline approach that executed planning with proposed proposition functions during validation, in Section VI.

3) Coarse Lexical Generation: To generate new lexical entries for words and phrases not present in the seed lexicon, we adapt the coarse lexical generation algorithm of [4] to our validation procedure. The algorithm generates new proposed lexicon entries from all possible combinations of factored lexical entries (see [16] for a detailed description of the lexicon) and words in a given training examples, then discards entries leading to parses that fail to validate.

V. Data Collection

Example Sentences Collected from AMT
"Move to the green room"
"Go by the red chair"
"Move to stand next to the chair"
"Move close to the chair"
"Stand in the blue room"

TABLE I: Example sentences collected from the AMT HIT where the users were shown a set of pre-and post condition states and asked to give a command that would instruct the robot to complete the task.

Our complete dataset (see Section V) contains 23 tasks, which include moving to any of the three rooms present in our domain, navigating to and relocating objects in the domain, all of which may be specified using room and object attributes, comparators ("the biggest room"), or in reference to relationships between objects in our domain ("go the room that the block is in"). Each task describes a set of goal states that the user desires the agent in our domain to reach. For each task, we generated five pre- and post- condition state pairs (Figure 2) using our simulator.

We then gathered training data using using the Amazon Mechanical Turk (AMT) platform. Users were shown three pairs of pre- and post- condition states, sampled from the total five, all representing the same task in different domain configurations. We chose the number of pairs to minimize cognitive load while maximizing the generalizability of the produced language. They were then asked to provide a single command that would instruct the robot to complete the task in every domain configuration. This provided multiple pre- and post- condition pairs for each training example, to incentivize both the learning algorithm and AMT users to produce outputs that are task- rather than configuration-specific. Some example commands gathered from AMT are shown in Table I. Using this approach, we collected a total of 2211 commands.

Many commands received from AMT users contained incorrect commands, many of which implied that the users only examined one of our three pairs of pre- and post- condition states. Some commands were vague and did not describe any specific task, while others only applied to the first pair of images shown to users. Including these commands in our data set actively misleads our parser, encouraging it to produce incorrect parses. We removed 174 such commands from the dataset to produce a pruned dataset.

VI. Experiments and Results

We performed three sets of experiments on our corpus, collected as described in Section V. First, we trained and tested our model on the full corpus collected from AMT, and a subset of the corpus manually pruned of incorrect and misleading commands. In the second experiment we tested our method against a baseline method that executes a planner during training, similar to earlier weakly supervised CCG parser learning approaches [4, 5]. This experiment was performed with a smaller training corpus, to allow planners with long horizons to validate the parser learning in a reasonable time-frame. For our third experiment, we reduced the level of supervision (that is, the number of demonstration state pairs,) available during learning, while leaving the test data unchanged.

Setup: We provided the learning algorithm with a seed lexicon as in [4, 5, 27], used to initialize the coarse lexical generation procedure described in Section IV-B.3. This seed lexicon contained words and phrases paired with syntactic categories and the lambda-calculus meaning representations elaborated upon in Section IV-A This seed lexicon contained simple noun phrases and adjectives such as "chair" and "red,", comparators such as "the largest", and imperatives such as "move to" or "go to," represented with corresponding lambda-calculus logical forms. In addition to using these lexical entries during parsing, the coarse lexical generation procedure used during learning produces new lexical entries from words found in the training data and meaning representations in the seed lexicon. We used a beam width of 75 for lexicon generation during training, which was performed over 15 epochs. At both training and test time, the CCG parser used a beam width of 150.

A. Evaluation on AMT Corpus

Training Set	Demos/Cmd	Best F1
Raw AMT	3	0.64
Pruned	3	**0.82**
Pruned	2	0.75
Pruned	1	0.71

TABLE II: F1 score on held-out test data, using varying levels of supervision and dataset quality.

We performed training on two permutations of the data set collected from AMT: the raw dataset, including many incorrect and misleading commands as described in Section V, and a pruned dataset with incorrect commands manually removed. Testing was performed by validating against state pairs provided with the held-out test data. The full dataset contained 1833 training commands and 678 test commands, each paired with 3 pre- and post- condition demonstration state pairs. The pruned dataset contained 1619 training commands and 418 test commands, each paired with 3 demonstrations as well. We observed a F1 score of 0.64 on the held-out test data of the raw dataset, and expected it to increase when errors were removed from the dataset as described in Section V. This pattern was observed, with an F1 score of 0.82 on the pruned dataset.

Planning horizon	Time taken to learn in s	F1 score
1 step	1201953 ($\sim$ 20 mins)	0.0
10 steps	24024701 ($\sim$ 6.67 hours)	0.667
20 steps	47597724 ($\sim$ 13.22 hours)	0.667
Our method (no planning)	87.18	0.72

TABLE III: Timing results and F1 scores on a dataset of a train and test split of 50 and 20 commands. The planning based parsing baseline spends most of the learning time computing plans for incorrect parses while learning to parse. The planning based methods spend more time learning and perform poorly when compared to our method.

B. Ablation Experiments

To test our parser's robustness to ambiguity in the demonstrations provided during learning, we performed learning with varying levels of supervision and tested against the same held-out test corpus. As our full corpus contains three pre- and post- condition state pairs as demonstration for each command, to incentivize the learner to produce compact and accurate proposition functions, we trained models with one and two commands provided as supervision and compared their performance. We hypothesized that providing fewer states as supervision would render the parser vulnerable to ambiguities in demonstrations - for example, if a provided command is "go near the block" but the learner is only provided with a demonstration in which the agent moves near the block but into the blue room, the proposition function $in(the(\lambda x.agent(x)), the(\lambda y.blue(y) \wedge y.room(y))$ would validate as correct, leading the parser to develop incorrect lexical entries and model weights. We observed that performance relative to the model provided with three demonstrations decreased progressively when demonstrations were removed, as shown in the final two rows of Table II.

C. Baselines

To compare against a planning-driven baseline, we created a dataset of 50 training and 20 test commands, containing 3 behaviors from our dataset. We produced full demonstration trajectories for these commands. We created the small dataset due to time constraints, as baseline methods take can upwards of 15 hours to run on a 50-command dataset. Much of the decrease in speed comes from attempting to validate invalid parses produced during training. The baseline method uses a planning based validation similar to that used in Artzi and Zettlemoyer [4]. In our baseline, plans are generated for every logical expression constructed and matched to demonstration trajectories in the training data. Parses are valid if the final state in the trajectory produced by the planner and the training trajectory satisfy the logical expression generated by the parser. Our method uses the first and final states of trajectories for parse validation during training, circumventing the repeated planning problem by ensuring that the logical expressions generated satisfy the terminal state of the dataset, without any planning. Using goal-state reward functions as task representations enables this simplification and the corresponding gain in efficiency.

The planning times can be made arbitrarily costly by increasing the horizon of planning, we present results for three different horizons, $1, 10, 15$, all of which take more time and preform worse than our method. The results are shown in Table III. The 1 step planning horizon is insufficient for the agent to reach the goal states in our dataset which can be $\sim 10-15$ steps away. However, 10 and 20 step planning horizons find correct parses with a comparable or slightly lower F1 score than our method.

VII. Robot Demonstration

To interface our parser with a Turtlebot robot, we use reward functions produced by the parser to produce plans in a physical mobile manipulation domain. The domain is a modeled as an MDP after Cleanup World [18]. The configuration of the three rooms can be changed. The states of the robot and the block are being tracked by a motion capture rig. The robot's actions are *forward*, *turn left*, *turn right*, which are executed on the robot as a series of Twist messages over ROS [22]. We show a user asking the robot to get to the right of the block, taking the block to the red room and moving to the largest room, and the robot performing these tasks appropriately. The video of the demonstrations is online at https://youtu.be/YChlga1wwAc.

When performing the task "get to the right of the block" the authors expected the agent to circle the block and thus arrive on its right side. However the planner computed that a more efficient plan would be to move the block so the agent reaches the goal of being to the right of the block. This behavior shows that there is more subtlety to language than our semantic representation captures - moving to the right of an object assumes that the object or the landmark has to be stationary. We do not model any landmarks or objects as stationary in our domain or our semantic representation, which can be added in future work. Moreover, this behavior also shows the importance of modeling tasks with MDPs so as to plan optimally.

Next we demonstrate the performance of our parser using limited training data and its ability to extend its model to tasks not seen at training time. We train our parser for two tasks: "go to the green room" and "move next to the block." We provide the parser with 6 sentences and a total of 6 pairs of pre- and post condition states. We collect this data from our Cleanup Domain simulator. Next we learn a parser for this data using our method using the parameters described in Section VI. We then use this parser to plan on the robot for the seen tasks of "going to the green room" and "going next to the block". More importantly, the parser successfully executes the unseen task of "going to the blue room." The seed lexicon provided to the parser during training contains the words: "blue", "green" and "red" along with their symbols *blue*, *green* and *red* respectively. The parser learns to infer that the symbols are adjectives used to describe objects and transfers this knowledge to unseen data, allowing the robot to plan for unseen commands with a very small amount of data. This small data set test shows the strength of our method to teach behavior to a robot that is generalizable with a handful annotations.

VIII. Discussion

In this work, we modeled goal-based tasks as reward functions within an MDP formalism. Mapping to reward functions allows the robot to plan optimally in real world stochastic environments. However, the tasks presented in this work are a subset of the possible tasks that humans can specify using natural language. For example tasks that specify a trajectory constraint such as "walk carefully along the river" cannot be modeled as a goal state. Mapping language to event based semantic representations such as those of Artzi and Zettlemoyer [4] would allow us to model such language. However, event based semantic representations requires tracking event history, which weakens the Markov assumption behind MDPs that enable efficient planning.

During data collection users described tasks using nested referential expressions in a way that we did not foresee. For example, users asked the agent to go the room that the chair was in, instead of asking the agent to go near the chair. Our learned parser composed nested referential expressions for these language commands to produce the right behavior while accurately representing the literal meaning of the language. However we noticed that our model had issues disambiguating between *in* and *near* symbols because the language that the users provided was similar when describing the tasks of going into a room and moving close to an object. For example: the phrase "go to" was used to command the robot to both enter a room and move near a block. We introduced new co-occurrence features to capture dependencies between pairs of predicates. This incentivized the parser to learn, for example, that an agent cannot get into a block. This leads to the correct behavior by the agent, but an incorrect representation of the sentences that use nested referential expressions in lambda calculus. This demonstrates that the parser's behavior can be altered in unexpected ways by the addition or removal of features.

Automated generation of reward functions from supervised or unsupervised methods is an important element of teaching behaviors to reinforcement learning or planning agents. We have focused on an approach that uses language and goal states to learn reward functions. In the weakly supervised setting Guu et al. [12] used algorithmic methods to reduce errors caused by ambiguity in demonstrations, which we instead address by providing more demonstrations to the parser to add clarity to the training commands provided. Other approaches have learned reward functions in an unsupervised setting from videos [23]. Future work could combine language and videos in an unsupervised setting.

IX. Conclusion

We presented a method for learning a parser that maps natural language commands to reward functions using a CCG parser via weak supervision. We showed that this parser can be learned using modifications of existing semantic parser learning algorithms, and its outputs are executable

as goal-state reward functions with off the shelf planners. Our model produces valid reward functions with an F1 score of 0.82 while showing an improvement in learning time over planning based methods. To the best of the authors' knowledge we are the first to use CCG parsing to produce MDP reward functions, enabling optimal planing in stochastic environments We also show the use of our learning method to learn from and generalize with a handful (6 instances) of training data. In future we would like to model more complicated temporal behaviors expressed using natural language, possibly with Linear Temporal Logic [21].

X. Acknowledgements

We thank Yoav Artzi for his insightful comments and help with the Cornell SPF library. We also thank George Konidaris and Lawson L.S. Wong for helpful feedback.

This work is supported by the National Science Foundation under grant number IIS-1637614, the US Army/DARPA under grant number W911NF-15-1-0503, and the National Aeronautics and Space Administration under grant number NNX16AR61G.

References

[1] Ken R. Anderson, Timothy J. Hickey, and Peter Norvig. The jscheme language and implementation, 2013. URL `http://jscheme.sourceforge.net/jscheme/main.html`.

[2] Yoav Artzi. Cornell SPF: Cornell Semantic Parsing Framework, 2016.

[3] Yoav Artzi and Luke Zettlemoyer. Bootstrapping semantic parsers from conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011.

[4] Yoav Artzi and Luke Zettlemoyer. Weakly supervized learning of semantic parsers for mapping instructions to actions. In *Annual Meeting of the Association for Computational Linguistics*, 2013.

[5] Yoav Artzi, Dipanjan Das, and Slav Petrov. Learning compact lexicons for ccg semantic parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1273–1283. Association for Computational Linguistics, October 2014.

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

[7] R. Bellman. A Markovian decision process. *Indiana University Mathematics Journal*, 6:679–684, 1957.

[8] B. Carpenter. *Type-Logical Semantics*. MIT Press, Cambridge, MA, USA, 1997.

[9] Stephen Clark and James R Curran. Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistics*, 33(4):493–552, 2007.

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

[11] Nakul Gopalan, Marie desJardins, Michael L. Littman, James MacGlashan, Shawn Squire, Stefanie Tellex, John Winder, and Lawson L. S. Wong. Planning with abstract markov decision processes. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling*, pages 480–488, 2017.

[12] 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.

[13] Thomas M. Howard, Stefanie Tellex, and Nicholas Roy. A natural language planner interface for mobile manipulators. In *IEEE International Conference on Robotics and Automation*, 2014.

[14] Michael Janner, Karthik Narasimhan, and Regina Barzilay. Representation learning for grounded spatial reasoning. *arXiv preprint arXiv:1707.03938*, 2017.

[15] Jayant Krishnamurthy and Tom M. Mitchell. Weakly supervised training of semantic parsers. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012.

[16] Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Lexical generalization in ccg grammar induction for semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1512–1523. Association for Computational Linguistics, 2011.

[17] James MacGlashan. Brown–UMBC Reinforcement Learning and Planning (BURLAP)–Project Page. http://burlap.cs.brown.edu/, 2014.

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

[19] Dipendra Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2017.

[20] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-707-2.

[21] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society. doi: 10.1109/SFCS.1977.32.

[22] Morgan Quigley, Josh Faust, Tully Foote, and Jeremy Leibs. Ros: an open-source robot operating system.

[23] Pierre Sermanet, Kelvin Xu, and Sergey Levine. Unsupervised perceptual rewards for imitation learning. *CoRR*, abs/1612.06699, 2016.

[24] Mark Steedman. *The Syntactic Process*. MIT Press, Cambridge, MA, USA, 2000. ISBN 0-262-19420-1.

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

[26] Edward C. Williams, Mina Rhee, Nakul Gopalan, and Stefanie Tellex. Learning to parse natural language to grounded reward functions with weak supervision. In *AAAI Fall Symposium on Natural Communication for Human-Robot Collaboration*, 2017.

[27] Luke Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2005.

[28] Luke Zettlemoyer and Michael Collins. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of the Joint Conference on Empirical Methods in Natural Lanugage Processing and Computational Natural Language Learning*, 2007.