Grounding Language for Physical Systems
AAAI Technical Report WS-12-07

Learning to Interpret Natural Language Instructions

James MacGlashan[*] and **Monica Babeş-Vroman**[+] and **Kevin Winner**[*] and **Ruoyuan Gao**[+] and **Richard Adjogah**[*] and **Marie desJardins**[*] and **Michael Littman**[+] and **Smaranda Muresan**[#]

[*] Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County
[+] Computer Science Department, Rutgers University
[#] School of Communication and Information, Rutgers University

Abstract

We address the problem of training an artificial agent to follow verbal commands using a set of instructions paired with demonstration traces of appropriate behavior. From this data, a mapping from instructions to tasks is learned, enabling the agent to carry out new instructions in novel environments. Our system consists of three components: semantic parsing (SP), inverse reinforcement learning (IRL), and task abstraction (TA). SP parses sentences into logical form representations, but when learning begins, the domain/task specific meanings of these representations are unknown. IRL takes demonstration traces and determines the likely reward functions that gave rise to these traces, defined over a set of provided features. TA combines results from SP and IRL over a set of training instances to create abstract goal definitions of tasks. TA also provides SP domain specific meanings for its logical forms and provides IRL the set of task-relevant features.

Introduction

Learning how to follow verbal instructions comes naturally to humans, but it has been a challenging task to automate. In this paper, we address the following problem: given a verbal instruction, what is the sequence of actions that the instructed agent needs to perform to successfully carry out the corresponding task? Such an agent is faced with many challenges: What is the meaning of the given sentence and how should it be parsed? How do words map to objects in the real world? Once the task is identified, how should the task be executed? Once the task is learned and executed, how can we generalize it to a new context, with objects and properties that the agent has not seen in the past? How should the agent's learning be evaluated?

The goal of our project is to develop techniques for a computer or robot to learn from examples to carry out multipart tasks, specified in natural language, on behalf of a user. Our approach uses three main subcomponents: (1) recognizing intentions from observed behavior using variations of Inverse Reinforcement Learning (IRL) methods; (2) translating instructions to task specifications using Semantic Parsing (SP) techniques; and (3) creating generalized task specifications to match user intentions, using probabilistic Task Abstraction (TA) methods.

In the next section, we discuss related work in this general problem domain. We then describe the architecture of our system and show results from a preliminary experiment. Finally, we summarize our ongoing and future research on this problem.

Related Work

Our work relates to the broad class of methods that aim to learn to interpret language from a situated context (Branavan et al. 2009; Branavan, Zettlemoyer, and Barzilay 2010; Branavan, Silver, and Barzilay 2011; Clarke et al. 2010; Chen and Mooney 2011; Vogel and Jurafsky 2010; Grubb et al. 2011; Goldwasser and Roth 2011; Liang, Jordan, and Klein 2011; Hewlett, Walsh, and Cohen 2010; Tellex et al. 2011; Atrzi and Zettlemoyer 2011). Instead of using annotated training data consisting of sentences and their corresponding logical forms (Kate and Mooney 2006; Wong and Mooney 2007; Zettlemoyer and Collins 2005; 2009), most of these approaches leverage non-linguistic information from a situated context as their primary source of supervision. These approaches have been applied to various tasks such as: interpreting verbal commands in the context of navigational instructions (Vogel and Jurafsky 2010; Chen and Mooney 2011; Grubb et al. 2011), robot manipulation (Tellex et al. 2011), puzzle solving and software control (Branavan et al. 2009; Branavan, Zettlemoyer, and Barzilay 2010); semantic parsing (Clarke et al. 2010; Liang, Jordan, and Klein 2011; Atrzi and Zettlemoyer 2011), learning game strategies from text (Branavan, Silver, and Barzilay 2011), and inducing knowledge about a domain based on text (Goldwasser and Roth 2011). The task closest to ours is interpreting navigation instructions. However, our goal is to

move away from low-level instructions that correspond directly to actions in the environment (Branavan et al. 2009; Vogel and Jurafsky 2010) to high-level task descriptions expressed using complex language.

Early work on grounded language learning used features based on n-grams to represent the natural language input (Branavan et al. 2009; Branavan, Zettlemoyer, and Barzilay 2010; Vogel and Jurafsky 2010). More recent methods have relied on a richer representation of linguistic data, such as syntactic dependency trees (Branavan, Silver, and Barzilay 2011; Goldwasser and Roth 2011) and semantic templates (Grubb et al. 2011; Tellex et al. 2011) to address the complexity of the natural language input. Our approach uses a flexible framework that allows us to incorporate various degrees of knowledge available at different stages in the learning process (e.g., from dependency relations to a full-fledged semantic model of the domain learned during training).

Background

We represent tasks using the Object-oriented Markov Decision Process (OO-MDP) formalism (Diuk, Cohen, and Littman 2008), an extension of Markov Decision Processes (MDPs) to explicitly capture relationships between objects. An MDP is a specific formulation of a decision-making problem that is defined by a four-tuple: $\{\mathcal{S}, \mathcal{A}, \mathcal{P}_{\cdot}(\cdot,\cdot)\mathcal{R}_{\cdot}(\cdot,\cdot)\}$, where $\mathcal{S}$ is the set of possible states in which the agent can find itself; $\mathcal{A}$ is the set of actions the agent can take; $\mathcal{P}_a(s, s')$ is the probability of the agent transitioning from state s to s' after applying action a in s; and $\mathcal{R}_a(s, s')$ is the reward received by the agent for transitioning from state s to s' after executing action a. In the context of defining a task corresponding to a particular goal, an MDP also defines a subset of states $\beta \subset \mathcal{S}$, called *termination states* that end an action sequence and goals are represented as terminal states that return a high reward.

The OO-MDP representation provides a structured factored state representation for MDPs. Specifically, OO-MDPs add a set of classes $\mathcal{C}$, each with a set of attributes $\mathcal{T}_{\mathcal{C}}$. Each OO-MDP state is defined by an unordered set of instantiated objects. In addition to these object definitions, an OO-MDP also defines a set of propositional functions. For instance, we might have a propositional function `toyIn(toy, room)` that operates on an object belonging to class "toy" and an object belonging to class "room," returning true if the specified "toy" object is in the specific "room" object. We extend OO-MDPs to include a set of propositional function classes ($\mathcal{F}$) associating propositional functions that describe similar properties.

An Example Domain

To illustrate our approach, we present a simple Sokoban-style domain called *Cleanup World*. Cleanup World is a 2D grid world defined by various rooms that are connected by open doorways. The world also contains various objects (toys) that the agent can move around to different positions in the world. The agent can move north, south, east, or west, unless a wall prevents the agent from doing so. To move a toy, the agent must be in an adjacent cell and move in the direction of the toy, resulting in the toy being pushed to the opposite adjacent cell from the agent (unless a wall or another toy is in the way, in which case nothing happens).

The Cleanup World domain can be represented as an OO-MDP with four object classes: agent, room, doorway, and toy. The agent class is defined by two attributes: the agent's x and y position in the world. It is assumed that every state includes exactly one agent object. The room class is defined by five attributes: the top, left, bottom, and right coordinates in the world defining a rectangle that spans the space of the room in the world (and implicitly defines a wall around this perimeter) and a color attribute that represents the color of the room. The doorway object is defined by x and y attributes representing its position in the world. We assume that doorways are only placed on the perimeter of a room wall, creating an opening in the doorway cell to allow the agent to enter or leave the room. The toy object is defined by three attributes: the toy's x and y position and an attribute specifying the toy's shape (star, moon, etc.). The OO-MDP for Cleanup World also defines a number of propositional functions that specify whether a toy is a specific shape (such as `isStar(toy)`), the color of a room (such as `isGreen(room)`), whether a toy is in a specific room (`toyIn(toy, room)`), and whether an agent is in a specific room (`agentIn(room)`). These functions belong to respective *shape*, *color*, and toy or agent *position* classes.

System Architecture

As mentioned in the introduction, the training data for the overall system is a set of pairs of verbal instructions and behavior. For example, one of these pairs could be the instruction *Move the star to the green room* with a demonstration of the task being accomplished in a specific environment containing various toys and rooms of different colors. We assume the availability of a set of features for each state represented using the OO-MDP propositional functions described previously. These features play an important role in defining the tasks to be learned. For example, a robot being taught to move toys around would have information about whether or not it is currently carrying a toy, what toy it needs to be moving, in which room it is currently, which room contains each toy, etc.

For each task, SP takes the verbal instruction and produces the corresponding logical form(s) (a form of dependency representation). If the semantics are unknown, then this parse will represent only the logical structure of the sentence. For example, if the robot receives the verbal instruction *Move the star to the green room*, SP will produce the logical form *move(*$star_1$*,*$room_1$*), P1(*$room_1$*, green)*. In this case, $star_1$ and $room_1$ are objects that the sentence references and there is an unknown action relationship ($move$) acting on $star_1$ and $room_1$ and an unknown relationship between the word "green" and the object $room_1$. After training, however, the semantics of this parse will be known and this sentence would be parsed as: *push(*$star_1$*,*$room_1$*), color(*$room_1$*, isGreen)*, where the sentence has been mapped to an internal task called $push$, and

the word green is mapped to the $isGreen$ propositional function belonging to class $color$.

Given a logical form, TA analyzes it to determine which task it could be referencing, creating a new task if no existing task seems relevant. TA then generates possible sets of propositional functions that would be relevant in the reward function of the tasks and passes them to IRL. Using the demonstration trajectory, IRL finds a reward function defined over the propositional functions in each set provided by TA, as well as the likelihood that the derived reward function is representative of what the agent did. Each reward function effectively represents the possible goals of the task by highlighting which propositional functions in a set would be relevant in that interpretation.

IRL returns the produced reward functions to TA, which uses the reward functions and their likelihoods to refine which task was intended from the sentence, as well as the semantics of the task (that is, how the task was applied, the relevant features of the task, and the task's definition of reward/termination functions). TA then sends this information to SP as an expression of the semantics of the logical form. This feedback in turn allows SP to learn the semantics of words in terms of OO-MDP propositional functions, enabling SP to produce logical forms with assigned semantics for future tasks, and allowing the system to derive tasks and reward functions from instructions in novel environments.

Inverse Reinforcement Learning

Inverse Reinforcement Learning (Abbeel and Ng 2004) addresses the task of learning a reward function from demonstrations of expert behavior and information about the state transition function. Recently, more data-efficient IRL methods have been proposed, including the Maximum Likelihood Inverse Reinforcement Learning (Babeş-Vroman et al. 2011) or MLIRL approach, which our system builds on. Given a small number of trajectories, MLIRL finds a weighing of the state features that (locally) maximizes the probability of these trajectories. In our system, these state features consist of one of the sets of propositional functions provided by TA. For a given task and a set of sets of state features, MLIRL evaluates the feature sets and returns to TA its assessment of the probabilities of the various sets.

Semantic Parsing

The Semantic Parsing component addresses the problem of mapping instructions to semantic parses, building on recent work on learning to map natural language utterances to meaning representations. The core of this approach is a recently developed grammar formalism, Lexicalized Well-Founded Grammar (LWFG), which balances expressiveness with practical—and provable—learnability results (Muresan and Rambow 2007; Muresan 2010; 2011). In LWFG, each string is associated with a syntactic-semantic representation, and the grammar rules have two types of constraints: one for semantic composition (Φ_c) and one for semantic interpretation (Φ_i). The semantic interpretation constraints, Φ_i, provide access to a semantic model (domain knowledge) during parsing. In the absence of a semantic model, however, the LWFG learnability result still holds. This fact is important if our agent is assumed to start with no knowledge of the task and domain. LWFG uses an ontology-based semantic representation, which is a logical form represented as a conjunction of atomic predicates. For example, the representation of the phrase *green room* is $\langle X_1.\textit{is=green}, X.P_1 = X_1, X.\textit{isa=room}\rangle$. The semantic representation specifies two concepts—green and room—connected through a property that can be uninstantiated in the absence of a semantic model, or instantiated via the Φ_i constraints to the property name (e.g, color) if such a model is present.

During the learning phase, the SP component, using an LWFG grammar that is learned offline, provides to TA the logical forms (i.e., the semantic parses, or the unlabeled dependency parses if no semantic model is given) for each verbal instruction. For example, for the instruction *Move the chair into the green room*, the SP component knows initially that *move* is a verb, *chair* and *room* are nouns, and *green* is an adjective. It also has grammar rules of the form S $\rightarrow$ Verb NP PP: Φ_{c1}, Φ_{i1},[1] but it has no knowledge of what these words mean (that is, to which concepts they map in the domain model). For this instruction, the LWFG parser returns the logical form $\langle (X_1.\textit{isa=move}, X_1.\textit{Arg1}= X_2)_{\textit{move}}, (X_2.\textit{det=the})_{\textit{the}}, (X_2.\textit{isa=chair})_{\textit{chair}}, (X_1.P_1 = X_3, P_2.\textit{isa=into})_{\textit{into}}, (X_3.\textit{det=the})_{\textit{the}}, (X_4.\textit{isa=green}, X_3.P_2 = X_2)_{\textit{green}}, (X_3.\textit{isa=room})_{\textit{room}}\rangle$. (The subscripts for each atomic predicate indicate the word to which that predicate corresponds.) This logical form corresponds to the simplified logical form `move(chair1,room1), P1(room1,green)`, where predicate P1 is uninstantiated. A key advantage of this framework is that the LWFG parser has access to the domain (semantic) model via Φ_i constraints. As a result, when TA provides feedback about domain-specific meanings (i.e., groundings), the parser can incorporate those mappings via the Φ_i constraints (e.g., move might map to the predicate "MoveToRoom" with a certain probability).

Task Abstraction

The termination conditions for an OO-MDP task can be defined in terms of the propositional functions. For example, the Cleanup World domain might include a task that requires the agent to put a specific toy (t_1) in a specific room (r_1). In this case, the termination states would be defined by states that satisfy $\texttt{toyIn}(t_1, r_1)$ and the reward function would be defined as $R_a(s, s') = \{1 : \texttt{toyIn}(t_1^{s'}, r_1^{s'}); -1 : \texttt{otherwise}\}$. However, such a task definition is overly specific and cannot be evaluated in a new environment that contains different objects. To remove this limitation, we define abstract task descriptions using *parametric lifted* reward and termination functions. A parametric lifted reward function is a first-order logic expression in which the propositional functions defining the reward can be selected as parameters. This representation allows much more general tasks to be defined; these tasks can be evaluated in any environment that contains the necessary object classes. For

[1]For readability, we show here just the context-free backbone, without the augmented nonterminals or constraints.

instance, the reward function for an abstract task that defines an agent taking a toy of a certain shape to a room of a certain color would be represented as $R_a(s, s') = \{1 : \exists_{t^{s'} \in toy} \exists_{r^{s'} \in room} \texttt{P1}(t) \wedge \texttt{P2}(r) \wedge \texttt{toyIn}(t, r); -1 : \texttt{otherwise}\}$, where `P1` is a propositional function that operates on toy objects (such as `isStar`) and `P2` is a propositional function that operates on room objects (such as `isGreen`). An analogous definition can be made for termination conditions.

Given the logical forms provided by SP, TA finds candidate tasks that might match each logical form, along with a set of possible *groundings* of those tasks. A grounding of an abstract task is the set of propositional functions (parameters of the abstract task) to be applied to the specific objects in a given training instance. TA then passes these grounded propositional functions as the features to use in IRL. (If there are no candidate tasks, then it will pass all grounded propositional functions of the OO-MDP to IRL.) When IRL returns a reward function for these possible groundings and their likelihoods of representing the true reward function, TA determines whether any abstract tasks it has defined might match. If not, TA will either create a new abstract task that is consistent with the received reward functions or it will modify one of its existing definitions, if doing so does not require significant changes. With IRL indicating the intended goal of a trace and with the abstract task indicating relevant parameters, TA can then inform SP of the semantics of its logical parse. The entire system proceeds iteratively, with each component, directly or indirectly, informing the others.

A Simplified System Example

In this section, we show a simplified version of our system with a unigram language model and minimal abstraction. We call this version Model 0. The input to Model 0 is as described: a set of verbal instructions paired with demonstrations of appropriate behavior. It uses the Expectation Maximization algorithm (Dempster, Laird, and Rubin 1977) to estimate the probability distribution of words conditioned on reward functions (the parameters). With this information, when the system receives a new command, it can behave in a way that maximizes its reward given the posterior probabilities of the possible reward functions given the words.

For all possible reward, demonstration pairs, the E-step of EM estimates $z_{ji} = \Pr(R_j|(S_i, T_i))$, the probability that reward function R_j produced sentence-trajectory pair (S_i, T_i). This estimate is given by the equation below:

$$\begin{aligned} z_{ji} &= \\ &= \Pr(R_j|(S_i,T_i)) = \frac{\Pr(R_j)}{\Pr(S_i,T_i)} \Pr((S_i,T_i)|R_j) \\ &= \frac{\Pr(R_j)}{\Pr(S_i,T_i)} \Pr(T_i|R_j) \Pr(S_i|R_j) \\ &= \frac{\Pr(R_j)}{\Pr(S_i,T_i)} \Pr(T_i|R_j) \prod_{w_k \in S_i} \Pr(w_k|R_j) \end{aligned} \quad (1)$$

where S_i is the ith sentence, T_i is the trajectory demonstrated for verbal command S_i, and w_k is an element in the set of all possible words (vocabulary).

Algorithm 1 EM Model 0

Input: Demonstrations $\{(S_1, T_1), ..., (S_N, T_N)\}$, number of reward functions J, size of vocabulary K.
Initialize: $x_{11}, \ldots, x_{JK}$, randomly.
repeat
E Step: Compute
$z_{ji} = \frac{\Pr(R_j)}{\Pr(S_i,T_i)} \Pr(T_i|R_j) \prod_{w_k \in S_i} x_{kj}$.

M step: Compute
$x_{kj} = \frac{1}{X} \frac{\Sigma_{w_k \in S_i} \Pr(R_j|S_i) + \epsilon}{\Sigma_i N(S_i) z_{ji} + \epsilon}$.
until target number of iterations completed.

If the reward functions R_j are known ahead of time, $\Pr(T_i|R_j)$ can be obtained directly by solving the MDP and estimating the probability of trajectory T_i under a Boltzmann policy with respect to R_j. If the R_js are not known, EM can estimate them by running IRL during the M-step (Babeş-Vroman et al. 2011).

The M-step uses the current estimates of z_{ji} to further refine the probabilities $x_{kj} = \Pr(w_k|R_j)$:

$$x_{kj} = \Pr(w_k|R_j) = \frac{1}{X} \frac{\Sigma_{w_k \in S_i} \Pr(R_j|S_i) + \epsilon}{\Sigma_i N(S_i) z_{ji} + \epsilon} \quad (2)$$

where ϵ is a smoothing parameter, X is a normalizing factor and $N(S_i)$ is the number of words in sentence S_i.

We show these steps in Algorithm 1.

Experiments

While our system is fully designed, it is still being implemented. As such, we only have results for specific parts of the system. We use these preliminary results to provide an illustration in the Cleanup World domain of how our final system will work. To collect a corpus of training data that would be linguistically interesting, we crowdsourced the task of generating instructions for example trajectories using Amazon Turk. Example trajectories were presented to users as an animated image of the agent interacting in the world, and users were asked to provide a corresponding instruction. This process had predictably mixed results: about 1/3 of the resulting instructions were badly malformed or inappropriate. For the results shown here, we have used "human-inspired" sentences, consisting of a manually constructed subset of sentences we received from our Turk experiment. These sentences were additionally simplified and clarified by retaining only the last verb and by pruning irrelevant portions of the sentence. Instructions are typically of the form, "Move the green star to the red room"; the trajectories in the training data consist of a sequence of states and actions that could be performed by the agent to achieve this goal.

Model 0

To illustrate the EM unigram model, we selected six random sentences for two tasks (three sentences for each task).

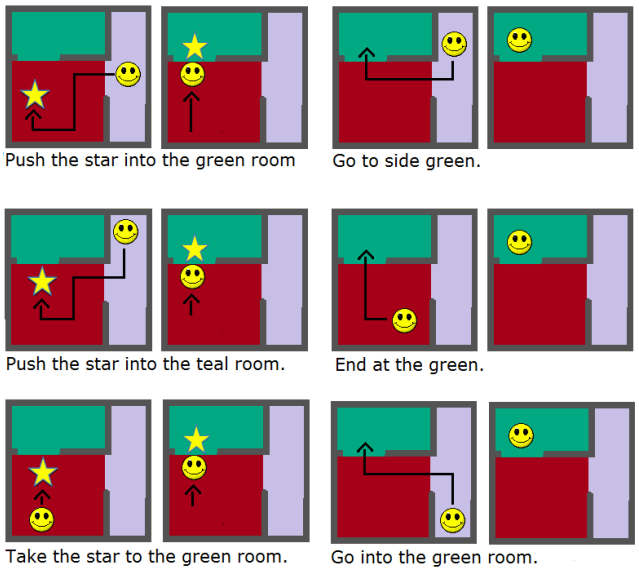

Figure 1: Training data for two tasks: 1) Taking the star to the green room (left) and 2) Going to the green room (right). Each training instance is a demonstration of the task (we show the starting state, the trace and the final state) paired with a verbal instruction, shown below the demonstration.

We show the training data in Figure 1. We obtained the reward function for each task using MLIRL, computed the $\Pr(T_i|R_j)$, then ran Algorithm 1 and obtained the parameters $\Pr(w_k|R_j)$.

After this training process, we presented the agent with a new task. She is given the instruction S_N: "Go to green room." and a starting state, somewhere in the same grid. Using parameters $\Pr(w_k|R_j)$, the agent can estimate:

$$\Pr(S_N|R_1) = \prod_{w_k \in S_N} \Pr(w_k|R_1) = 8.6 \times 10^{-7},$$

$$\Pr(S_N|R_2) = \prod_{w_k \in S_N} \Pr(w_k|R_2) = 4.1 \times 10^{-4}$$

and choose the optimal policy corresponding to reward R_2, thus successfully carrying out the task. Note that R_1 and R_2 corresponded to the two target tasks, but this mapping was determined by EM.

Using a minimalistic abstraction model, the Model 0 agent could learn that words like "green" and "teal" map to the same abstract color. Still, this agent is very limited and could benefit from a richer abstraction and a language model that can capture semantic information. Our aim is to have an agent learn what words mean and how they map to objects and their attributes in her environment.

We illustrate the limitation of the unigram model by telling the trained agent to "Go with the star to green." (we label this sentence S'_N). Using the learned parameters, the agent will compute the following estimates:

$$\Pr(S'_N|R_1) = \prod_{w_k \in S'_N} \Pr(w_k|R_1) = 8.25 \times 10^{-7},$$

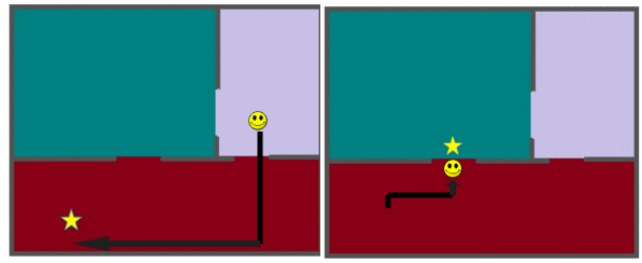

Figure 2: A task demonstration example for the verbal instruction "Push the star into the green room".

$$\Pr(S'_N|R_2) = \prod_{w_k \in S'_N} \Pr(w_k|R_2) = 2.10 \times 10^{-5}.$$

The agent wrongly chooses reward R_2 and goes to the green room instead of taking the star to the green room. The problem with the unigram model in this case is that it gives too much weight to word frequencies (in this case "go") without taking into account what the words mean or how they are used in the context of the sentence.

We try to address some of these issues in Model 1 by bringing to bear more complex language and abstraction models.

Model 1

We assume that SP has the ability to take a verbal command and interpret it as one or more possible logical forms. We also assume that the OO-MDP definition of the domain includes a large set of propositional functions capable of defining the target task features. The IRL component can take a set of features and a task demonstration and assign the feature weights that make this demonstration the most likely. The role of the TA component is to put this information together and learn abstract tasks.

When learning about a command, the amount of existing information may vary. In this section, we illustrate the learning process 1) when almost all semantic information has already been learned, 2) when a new command is encountered, but existing task knowledge is available, and 3) when there is no pre-existing information. We will use as an example, the simple task shown in Figure 2 with the verbal command "Push the star into the green room."

Learning a New Color Name

The easiest learning challenge is when the semantics of a command are fully known except for one word. In this example, we assume that the semantics of all the words in the command "Push the star into the green room" are known except for the word "green."

SP produces the logical parse:
PushToRoom(star, room), P1(room, green), shape(star, isStar). Since the task of pushing an object to a differently colored room has already been learned, the logical parse contains semantic knowledge to indicate that the command refers to the task "PushToRoom" and that the noun "star" indicates a star–shaped object, but SP does not know what the

adjective modifier "green" means, in which case it can only identify that a logical relation between the word "green" and the object $room_1$ exists, not what it means.

Provided with this incomplete semantic parse, TA finds all the possible and consistent groundings of the "PushToRoom(room, toy)" abstract task for the corresponding trajectory. From the semantic parse, it is also known that the toy must satisfy the `isStar` proposition and because the initial state of the demonstration only contains one star toy, the scope of possible groundings is limited to those referring it. The only unknown is the intended room. Therefore, TA considers the groundings of "PushToRoom" for every one of the three rooms in the environment:

- PushToRoom(t_1,r_1), isStar(t_1)
- PushToRoom(t_1,r_2), isStar(t_1)
- PushToRoom(t_1,r_3), isStar(t_1)

Each of these groundings of "PushToRoom" induces the relevant feature set sent to IRL. For instance, the `toyIn`(toy,room) feature required by "PushToRoom" has the three grounded versions `toyIn`(t_1,r_1), `toyIn`(t_1,r_2) and `toyIn`(t_1,r_3).

IRL's predicted reward function over the features indicates the correct grounding; for instance, since the agent takes the toy to the green room in the trajectory, the `toyIn` function grounded to the green room object will have a higher weight than the other groundings. Any propositional function operating on the identified grounded room object is sent back to SP as a possible semantic interpretation for $P1$, such as $color(room_1, isGreen)$, since the room satisfies $isGreen$, which belongs to class $color$.

Processing a New Command

Let us now consider the case when all the words in the verbal command are unknown, but the corresponding abstract task is known. In this case, SP parses the sentence into the following possible logical forms without semantic knowledge:

1. push(star,room), P1(room,green) (L_1)
2. push(star), P1(room,green), prep_into(star,room) (L_2)

The score associated with each of the possible parses represents the likelihood that the parse is correct. Here, L_1 corresponds to [push [the star] [into the green room]] and L_2 to [push [the star into the green room]]. We normalize these probabilities and then compute the probability of each logical form by summing up the normalized probabilities of the parses. In this example, we obtain $\Pr(L_1) = 0.65$ and $\Pr(L_2) = 0.35$.

These logical forms and their scores are passed to TA. Based on the content of these parses, TA selects likely candidates for the abstracted task, among the tasks that have already been learned. For instance, the possible tasks selected by TA might be "PushToRoom," "MoveToRoom," and "MoveToObject." TA then populates a set of relevant features based on these tasks and gives this list to IRL.

IRL processes these sets of features using the MLIRL algorithm to compute a set of weights for the features in each set. IRL returns to TA a collection of these weighted features. We show some of the ones with the highest weights:
$(1.91, \texttt{toyIn}(t_1, r_1))$,
$(1.12, \texttt{agentTouchingToy}(t_1))$,
$(0.80, \texttt{agentIn}(r_1))$.

With the information received from the SP and IRL components, TA now has the task of refining the grounded reward functions from the abstract tasks. TA then provides this grounding information to SP, which updates its semantic representations. The entire process is then repeated in an iterative-improvement fashion until a stopping condition is met.

Learning from Scratch

Our system is trained using a set $((S_1, T_1), ..., (S_N, T_N))$, of sentence–trajectory pairs. Initially, the system does not know what any of the words mean and there are no preexisting abstract tasks. Here we consider what happens when the system sees a task for the first time.

Let's assume that S_1 is "Push the star into the green room." This sentence is first processed by SP, yielding the following parses:

$$\Pr(L_1) = 0.65$$
$$\Pr(L_2) = 0.35$$

where L_1 is push(star,room), P1(room,green) and L_2 is push(star), P1(room,green), prep_into(star,room).

These parses and their likelihoods are passed to TA, and TA induces incomplete abstract tasks, which define only the number and kinds of objects that are relevant to the corresponding reward function. TA can send to IRL a set of features involving these objects, together with T_1, the demonstration attached to S_1. This set of features might include: $\texttt{toyIn}(t_1, r_1), \texttt{toyIn}(t_1, r_2), \texttt{agentTouchingToy}(t_1)$, $\texttt{agentIn}(r_1)$.

IRL sends back a weighting of the features, and TA can select the subset of features that have the highest weights, for example:
$(1.91, \texttt{toyIn}(t_1, r_1))$,
$(1.12, \texttt{agentTouchingToy}(t_1))$,
$(0.80, \texttt{agentIn}(r_1))$.

Using information from SP and IRL, TA can now create a new abstract task, called for example "PushToRoom", adjust the probabilities of the parses based on the relevant features obtained from IRL, and send these probabilities back to SP, enabling it to adjust its semantic model.

Conclusions and Future Work

Our project grounds language in a simulated environment by training an agent from verbal commands paired with demonstration of appropriate behavior. We decompose the problem into three major modules and show how iteratively exchanging information between these modules can result in correct interpretations of commands. We show some of the shortcomings of our initial unigram language model and propose an SP component that includes the possibility of building an ontology. Another crucial component we add is a component in charge of building abstract tasks from language information and feature relevance. We believe that learning

semantics will enable our system to carry out commands that the system has not seen before.

References

Abbeel, P., and Ng, A. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference in Machine Learning (ICML 2004)*.

Atrzi, Y., and Zettlemoyer, L. 2011. Bootstrapping semantic parsers for conversations. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*.

Babeş-Vroman, M.; Marivate, V.; Subramanian, K.; and Littman, M. 2011. Apprenticeship learning about multiple intentions. In *Proceedings of the Twenty Eighth International Conference on Machine Learning (ICML 2011)*.

Branavan, S. R. K.; Chen, H.; Zettlemoyer, L. S.; and Barzilay, R. 2009. Reinforcement learning for mapping instructions to actions. In *Association for Computational Linguistics (ACL 2009)*.

Branavan, S.; Silver, D.; and Barzilay, R. 2011. Learning to win by reading manuals in a monte-carlo framework. In *Association for Computational Linguistics (ACL 2011)*.

Branavan, S. R. K.; Zettlemoyer, L. S.; and Barzilay, R. 2010. Reading between the lines: Learning to map high-level instructions to commands. In *Association for Computational Linguistics (ACL 2010)*.

Chen, D. L., and Mooney, R. J. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-2011)*., 859–865.

Clarke, J.; Goldwasser, D.; Chang, M.-W.; and Roth, D. 2010. Driving semantic parsing from the world's response. In *Proceedings of the Association for Computational Linguistics (ACL 2010)*.

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society* 39(1):1–38.

Diuk, C.; Cohen, A.; and Littman, M. 2008. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-08)*.

Goldwasser, D., and Roth, D. 2011. Learning from natural instructions. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*.

Grubb, A.; Duvallet, F.; Tellex, S.; Kollar, T.; Roy, N.; Stentz, A.; and Bagnel, J. A. 2011. Imitation learning for natural language direction following. In *Proceedings of the ICML Workshop on New Developments in Imitation Learning*.

The authors acknowledge the support of the National Science Foundation (collaborative grant IIS-00006577 and IIS-1065195). Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors, and do not necessarily reflect the views of the funding organization.

Hewlett, D.; Walsh, T. J.; and Cohen, P. R. 2010. Teaching and executing verb phrases. In *Proceedings of the First Joint IEEE International Conference on Development and Learning and on Epigenetic Robotics (ICDL-Epirob-11)*.

Kate, R. J., and Mooney, R. J. 2006. Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44.

Liang, P.; Jordan, M.; and Klein, D. 2011. Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL 2011)*.

Muresan, S., and Rambow, O. 2007. Grammar approximation by representative sublanguage: A new model for language learning. In *Proceedings of ACL*.

Muresan, S. 2010. A learnable constraint-based grammar formalism. In *Proceedings of COLING*.

Muresan, S. 2011. Learning for deep language understanding. In *Proceedings of IJCAI-11*.

Tellex, S.; Kollar, T.; Dickerson, S.; Walter, M.; Banerjee, A. G.; Teller, S.; and Roy, N. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the Twenty-Fifth AAAI Conference on Articifical Intelligence*.

Vogel, A., and Jurafsky, D. 2010. Learning to follow navigational directions. In *Association for Computational Linguistics (ACL 2010)*.

Wong, Y. W., and Mooney, R. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-2007)*.

Zettlemoyer, L. S., and Collins, M. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of UAI-05*.

Zettlemoyer, L., and Collins, M. 2009. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Association for Computational Linguistics (ACL'09)*.