Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions

Yoav Artzi and **Luke Zettlemoyer**
Computer Science & Engineering
University of Washington
Seattle, WA 98195
{yoav,lsz}@cs.washington.edu

Abstract

The context in which language is used provides a strong signal for learning to recover its meaning. In this paper, we show it can be used within a grounded CCG semantic parsing approach that learns a joint model of meaning and context for interpreting and executing natural language instructions, using various types of weak supervision. The joint nature provides crucial benefits by allowing situated cues, such as the set of visible objects, to directly influence learning. It also enables algorithms that learn while executing instructions, for example by trying to replicate human actions. Experiments on a benchmark navigational dataset demonstrate strong performance under differing forms of supervision, including correctly executing 60% more instruction sets relative to the previous state of the art.

1 Introduction

The context in which natural language is used provides a strong signal to reason about its meaning. However, using such a signal to automatically learn to understand unrestricted natural language remains a challenging, unsolved problem.

For example, consider the instructions in Figure 1. Correct interpretation requires us to solve many subproblems, such as resolving all referring expressions to specific objects in the environment (including, "the corner" or "the third intersection"), disambiguating word sense based on context (e.g., "the chair" could refer to a chair or sofa), and finding executable action sequences that satisfy stated constraints (such as "twice" or "to face the blue hall").

move forward twice to the chair
$\lambda a.move(a) \wedge dir(a, forward) \wedge len(a, 2) \wedge$
$to(a, \iota x.chair(x))$
at the corner turn left to face the blue hall
$\lambda a.pre(a, \iota x.corner(x)) \wedge turn(a) \wedge dir(a, left) \wedge$
$post(a, front(you, \iota x.blue(x) \wedge hall(x)))$
move to the chair in the third intersection
$\lambda a.move(a) \wedge to(a, \iota x.sofa(x)) \wedge$
$intersect(order(\lambda y.junction(y), frontdist, 3), x)$

Figure 1: A sample navigation instruction set, paired with lambda-calculus meaning representations.

We must also understand implicit requests, for example from the phrase "at the corner," that describe goals to be achieved without specifying the specific steps. Finally, to do all of this robustly without prohibitive engineering effort, we need *grounded* learning approaches that jointly reason about meaning and context to learn directly from their interplay, with as little human intervention as possible.

Although many of these challenges have been studied separately, as we will review in Section 3, this paper represents, to the best of our knowledge, the first attempt at a comprehensive model that addresses them all. Our approach induces a weighted Combinatory Categorial Grammar (CCG), including both the parameters of the linear model and a CCG lexicon. To model complex instructional language, we introduce a new semantic modeling approach that can represent a number of key linguistic constructs that are common in spatial and instructional language. To learn from indirect supervision, we define the notion of a validation function, for example that tests the state of the agent after interpreting an instruction. We then show how this function can be used to drive online learning. For

Transactions of the Association for Computational Linguistics, 1 (2013) 49–62. Action Editor: Jason Eisner.
Submitted 11/2012; Published 3/2013.

that purpose, we adapt the loss-sensitive Perceptron algorithm (Singh-Miller & Collins, 2007; Artzi & Zettlemoyer, 2011) to use a validation function and coarse-to-fine inference for lexical induction.

The joint nature of this approach provides crucial benefits in that it allows situated cues, such as the set of visible objects, to directly influence parsing and learning. It also enables the model to be learned while executing instructions, for example by trying to replicate actions taken by humans. In particular, we show that, given only a small seed lexicon and a task-specific executor, we can induce high quality models for interpreting complex instructions.

We evaluate the method on a benchmark navigational instructions dataset (MacMahon et al., 2006; Chen & Mooney, 2011). Our joint approach successfully completes 60% more instruction sets relative to the previous state of the art. We also report experiments that vary supervision type, finding that observing the final position of an instruction execution is nearly as informative as observing the entire path. Finally, we present improved results on a new version of the MacMahon et al. (2006) corpus, which we filtered to include only executable instructions paired with correct traces.

2 Technical Overview

Task Let $\mathcal{S}$ be the set of possible environment states and $\mathcal{A}$ be the set of possible actions. Given a start state $s \in \mathcal{S}$ and a natural language instruction x, we aim to generate a sequence of actions $\vec{a} = \langle a_1, \ldots, a_n \rangle$, with each $a_i \in \mathcal{A}$, that performs the steps described in x.

For example, in the navigation domain (MacMahon et al., 2006), $\mathcal{S}$ is a set of positions on a map. Each state $s = (x, y, o)$ is a triple, where x and y are integer grid coordinates and $o \in \{0, 90, 180, 270\}$ is an orientation. Figure 2 shows an example map with 36 states; the ones we use in our experiments contain an average of 141. The space of possible actions $\mathcal{A}$ is $\{\text{LEFT}, \text{RIGHT}, \text{MOVE}, \text{NULL}\}$. Actions change the state of the world according to a transition function $T : \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{S}$. In our navigation example, moving forward can change the x or y coordinates while turning changes the orientation o.

Model To map instructions to actions, we jointly reason about linguistic meaning and action execution. We use a weighted CCG grammar to rank possible meanings z for each instruction x. Section 6 defines how to design such grammars for instructional language. Each logical form z is mapped to a sequence of actions $\vec{a}$ with a deterministic executor, as described in Section 7. The final *grounded CCG* model, detailed in Section 6.3, jointly constructs and scores z and $\vec{a}$, allowing for robust situated reasoning during semantic interpretation.

Learning We assume access to a training set containing n examples $\{(x_i, s_i, \mathcal{V}_i) : i = 1 \ldots n\}$, each containing a natural language sentence x_i, a start state s_i, and a validation function $\mathcal{V}_i$. The validation function $\mathcal{V}_i : A \rightarrow \{0, 1\}$ maps an action sequence $\vec{a} \in A$ to 1 if it's correct according to available supervision, or 0 otherwise. This training data contains no direct evidence about the logical form z_i for each x_i, or the grounded CCG analysis used to construct z_i. We model all these choices as latent variables. We experiment with two validation functions. The first, $\mathcal{V}^D(\vec{a})$, has access to an observable demonstration of the execution $\vec{a}_i$, a given $\vec{a}$ is valid iff $\vec{a} = \vec{a}_i$. The second, $\mathcal{V}_i^S(\vec{a})$, only encodes the final state s_i' of the execution of x, therefore $\vec{a}$ is valid iff its final state is s_i'. Since numerous logical forms often execute identically, both functions provide highly ambiguous supervision.

Evaluation We evaluate task completion for single instructions on a test set $\{(x_i, s_i, s_i') : i = 1 \ldots n\}$, where s_i' is the final state of an oracle agent following the execution of x_i starting at state s_i. We will also report accuracies for correctly interpreting instruction sequences $\vec{x}$, where a single error can cause the entire sequence to fail. Finally, we report accuracy on recovering correct logical forms z_i on a manually annotated subset of the test set.

3 Related Work

Our learning is inspired by the reinforcement learning (RL) approach of Branavan et al. (2009), and related methods (Vogel & Jurafsky, 2010), but uses latent variable model updates within a semantic parser. Branavan et al. (2010) extended their RL approach to model high-level instructions, which correspond to implicit actions in our domain. Wei et al. (2009) and Kollar et al. (2010) used shallow linguistic representations for instructions. Recently, Tellex

et al. (2011) used a graphical model semantics representation to learn from instructions paired with demonstrations. In contrast, we model significantly more complex linguistic phenomena than these approaches, as required for the navigation domain.

Other research has adopted expressive meaning representations, with differing learning approaches. Matuszek et al. (2010, 2012) describe supervised algorithms that learn semantic parsers for navigation instructions. Chen and Mooney (2011), Chen (2012) and Kim and Mooney (2012) present state-of-the-art algorithms for the navigation task, by training a supervised semantic parser from automatically induced labels. Our work differs in the use of joint learning and inference approaches.

Supervised approaches for learning semantic parsers have received significant attention, e.g. Kate and Mooney (2006), Wong and Mooney (2007), Muresan (2011) and Kwiatkowski et al. (2010, 2012). The algorithms we develop in this paper combine ideas from previous supervised CCG learning work (Zettlemoyer & Collins, 2005, 2007; Kwiatkowski et al., 2011), as we describe in Section 4. Recently, various alternative forms of supervision were introduced. Clarke et al. (2010), Goldwasser and Roth (2011) and Liang et al. (2011) describe approaches for learning semantic parsers from sentences paired with responses, Krishnamurthy and Mitchell (2012) describe using distant supervision, Artzi and Zettlemoyer (2011) use weak supervision from conversational logs and Goldwasser et al. (2011) present work on unsupervised learning. We discuss various forms of supervision that complement these approaches. There has also been work on learning for semantic analysis tasks from grounded data, including event streams (Liang et al., 2009; Chen et al., 2010) and language paired with visual perception (Matuszek et al., 2012).

Finally, the topic of executing instructions in non-learning settings has received significant attention (e.g., Winograd (1972), Di Eugenio and White (1992), Webber et al. (1995), Bugmann et al. (2004), MacMahon et al. (2006) and Dzifcak et al. (2009)).

4 Background

We use a weighted linear CCG grammar for semantic parsing, as briefly reviewed in this section.

Combinatory Categorial Grammars (CCGs) CCGs are a linguistically-motivated formalism for modeling a wide range of language phenomena (Steedman, 1996, 2000). A CCG is defined by a lexicon and a set of combinators. The lexicon contains entries that pair words or phrases with categories. For example, the lexical entry $\text{chair} \vdash N : \lambda x.chair(x)$ for the word "chair" in the parse in Figure 4 pairs it with a category that has syntactic type N and meaning $\lambda x.chair(x)$. Figure 4 shows how a CCG parse builds a logical form for a complete sentence in our example navigation domain. Starting from lexical entries, each intermediate parse node, including syntax and semantics, is constructed with one of a small set of CCG combinators (Steedman, 1996, 2000). We use the application, composition and coordination combinators, and three others described in Section 6.3.

Factored CCG Lexicons Recently, Kwiatkowski et al. (2011) introduced a factored CCG lexicon representation. Each lexical item is composed of a lexeme and a template. For example, the entry $\text{chair} \vdash N : \lambda x.chair(x)$ would be constructed by combining the lexeme $\text{chair} \vdash [chair]$, which contains a word paired with logical constants, with the template $\lambda v.[N : \lambda x.v(x)]$, that defines the rest of the category by abstracting over logical constants. This approach allows the reuse of common syntactic structures through a small set of templates. Section 8 describes how we learn such lexical entries.

Weighted Linear CCGs A weighted linear CCG (Clark & Curran, 2007) ranks the space of possible parses under the grammar, and is closely related to several other approaches (Lafferty et al., 2001; Collins, 2004; Taskar et al., 2004). Let x be a sentence, y be a CCG parse, and $\text{GEN}(x; \Lambda)$ be the set of all possible CCG parses for x given the lexicon Λ. Define $\phi(x, y) \in \mathbb{R}^d$ to be a d-dimensional *feature–vector* representation and $\theta \in \mathbb{R}^d$ to be a parameter vector. The optimal parse for sentence x is

$$y^*(x) = \arg \max_{y \in \text{GEN}(x;\Lambda)} \theta \cdot \phi(x, y)$$

and the final output logical form z is the λ-calculus expression at the root of $y^*(x)$. Section 7.2 describes how we efficiently compute an approximation to $y^*(x)$ within the joint interpretation and execution model.

Supervised learning with GENLEX Previous work (Zettlemoyer & Collins, 2005) introduced a function GENLEX(x, z) to map a sentence x and its meaning z to a large set of potential lexical entries. These entries are generated by rules that consider the logical form z and guess potential CCG categories. For example, the rule $p \rightarrow N : \lambda x.p(x)$ introduces categories commonly used to model certain types of nouns. This rule would, for example, introduce the category $N : \lambda x.chair(x)$ for any logical form z that contains the constant $chair$. GENLEX uses a small set of such rules to generate categories that are paired with all possible substrings in x, to create a large set of lexical entries. The complete learning algorithm then simultaneously selects a small subset of these entries and estimates parameter values θ. In Section 8, we will introduce a new way of using GENLEX to learn from different signals that, crucially, do not require a labeled logical form z.

5 Spatial Environment Modeling

We will execute instructions in an environment, see Section 2, which has a set of positions. A position is a triple (x, y, o), where x and y are horizontal and vertical coordinates, and $o \in O = \{0, 90, 180, 270\}$ is an orientation. A position also includes properties indicating the object it contains, its floor pattern and its wallpaper. For example, the square at $(4, 3)$ in Figure 2 has four positions, one per orientation.

Because instructional language refers to objects and other structures in an environment, we introduce the notion of a position set. For example, in Figure 2, the position set $D = \{(5, 3, o) : o \in O\}$ represents a chair, while $B = \{(x, 3, o) : o \in O, x \in [0 \dots 5]\}$ represents the blue floor. Both sets contain all orientations for each (x, y) pair, thereby representing properties of regions. Position sets can have many properties. For example, E, in addition to being a chair, is also an intersection because it overlaps with the neighboring halls A and B. The set of possible entities includes all position sets and a few additional entries. For example, set $C = \{(4, 3, 90)\}$ in Figure 2 represents the agent's position.

6 Modeling Instructional Language

We aim to design a semantic representation that is learnable, models grounded phenomena such as spatial relations and object reference, and is executable.

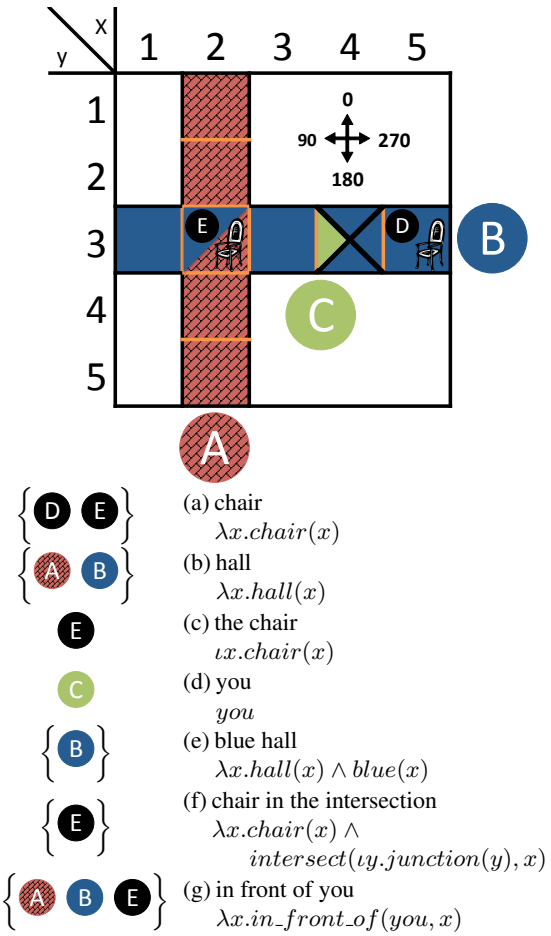


Figure 2: Schematic diagram of a map environment and example of semantics of spatial phrases.

Our semantic representation combines ideas from Carpenter (1997) and Neo-Davidsonian event semantics (Parsons, 1990) in a simply typed λ-calculus. There are four basic types: (1) entities e that are objects in the world, (2) events ev that specify actions in the world, (3) truth values t, and (4) meta-entities m, such as numbers or directions. We also allow functional types, which are defined by input and output types. For example, $\langle e, t \rangle$ is the type of function from entities to truth values.

6.1 Spatial Language Modeling

Nouns and Noun Phrases Noun phrases are paired with e-type constants that name specific entities and nouns are mapped to $\langle e, t \rangle$-type expressions that define a property. For example, the noun "chair" (Figure 2a) is paired with the expression $\lambda x.chair(x)$, which defines the set of objects for

which the constant $chair$ returns true. The denotation of this expression is the set $\{D, E\}$ in Figure 2 and the denotation of $\lambda x.hall(x)$ (Figure 2b) is $\{A, B\}$. Also, the noun phrase "you" (Figure 2d), which names the agent, is represented by the constant you with denotation C, the agent's position.

Determiners Noun phrases can also be formed by combining nouns with determiners that pick out specific objects in the world. We consider both definite reference, which names contextually unique objects, and indefinites, which are less constrained.

The definite article is paired with a logical expression ι of type $\langle\langle e, t\rangle, e\rangle$,[1] which will name a single object in the world. For example, the phrase "the chair" in Figure 2c will be represented by $\iota x.chair(x)$ which will denote the appropriate chair. However, computing this denotation is challenging when there is perceptual ambiguity, for positions where multiple chairs are visible. We adopt a simple heuristic approach that ranks referents based on a combination of their distance from the agent and whether they are in front of it. For our example, from position C our agent would pick the chair E in front of it as the denotation. The approach differs from previous, non-grounded models that fail to name objects when faced with such ambiguity (e.g., Carpenter (1997), Heim and Kratzer (1998)).

To model the meaning of indefinite articles, we depart from the Frege-Montague tradition of using existential quantifiers (Lewis, 1970; Montague, 1973; Barwise & Cooper, 1981), and instead introduce a new quantifier $\mathcal{A}$ that, like ι, has type $\langle\langle e, t\rangle, e\rangle$. For example, the phrase "a chair" would be paired with $\mathcal{A}x.chair(x)$ which denotes an arbitrary entry from the set of chairs in the world. Computing the denotation for such expressions in a world will require picking a specific object, without further restrictions. This approach is closely related to Steedman's generalized Skolem terms (2011).[2]

Meta Entities We use m-typed terms to represent non-physical entities, such as numbers (1, 2, etc.) and directions ($left$, $right$, etc.) whose denotations are fixed. The ability to refer to directions allows us to manipulate position sets. For example, the phrase "your left" is mapped to the logical expression $orient(you, left)$, which denotes the position set containing the position to the left of the agent.

Prepositions and Adjectives Noun phrases with modifiers, such as adjectives and prepositional phrases are $\langle e, t\rangle$-type expressions that implement set intersection with logical conjunctions. For example in Figure 2, the phrase "blue hall" is paired with $\lambda x.hall(x) \wedge blue(x)$ with denotation $\{B\}$ and the phrase "chair in the intersection" is paired with $\lambda x.chair(x) \wedge intersect(\iota y.junction(y), x)$ with denotation $\{E\}$. Intuitively, the adjective "blue" introduces the constant $blue$ and "in the" adds a $intersect$. We will describe the full details of how these expressions are constructed in Section 6.3.

Spatial Relations The semantic representation allows more complex reasoning over position sets and the relations between them. For example, the binary relation in_front_of (Figure 2g) tests if the first argument is in front of the second from the point of view of the agent. Additional relations are used to model set intersection, relative direction, relative distance, and relative position by distance.

6.2 Modeling Instructions

To model actions in the world, we adopt Neo-Davidsonian event semantics (Davidson, 1967; Parsons, 1990), which treats events as ev-type primitive objects. Such an approach allows for a compact lexicon where adverbial modifiers introduce predicates, which are linked by a shared event argument.

Instructional language is characterized by heavy usage of imperatives, which we model as functions from events to truth values.[3] For example, an imperative such as "move" would have the meaning $\lambda a.move(a)$, which defines a set of events that match the specified constraints. Here, this set would include all events that involve moving actions.

The denotation of ev-type terms is a sequence of n instances of the same action. In this way, an event defines a function $ev : s \rightarrow s'$, where s is the start state and s' the end state. For example, the

[1] Although quantifiers are logical constants with type $\langle\langle e, t\rangle, e\rangle$ or $\langle\langle e, t\rangle, t\rangle$, we use a notation similar to that used for first-order logic. For example, the notation $\iota x.f(x)$ represents the logical expression $\iota(\lambda x.f(x))$

[2] Steedman (2011) uses generalized Skolem terms as a tool for resolving anaphoric pronouns, which we do not model.

[3] Imperatives are $\langle ev, t\rangle$-type, much like $\langle e, t\rangle$-type wh-interrogatives. Both define sets, the former includes actions to execute, the later defines answers to a question.

denotation of $\lambda a.move(a)$ is the set of move action sequences $\{\langle \text{MOVE}_1, \ldots, \text{MOVE}_n \rangle : n \geq 1\}$. Although performing actions often require performing additional ones (e.g., the agent might have to $turn$ before being able to $move$), we treat such actions as implicit (Section 7.1), and don't model them explicitly within the logical form.

Predicates such as $move$ (seen above) and $turn$ are introduced by verbs. Events can also be modified by adverbials, which are intersective, much like prepositional phrases. For example in the imperative, logical form (LF) pair:

Imp.: move from the sofa to the chair
LF: $\lambda a.move(a) \wedge to(a, \iota x.chair(x)) \wedge from(a, \iota y.sofa(y))$

Each adverbial phrase provides a constraint, and changing their order will not change the LF.

6.3 Parsing Instructional Language with CCG

To compose logical expressions from sentences we use CCG, as described in Section 4. Figures 3 and 4 present a sample of lexical entries and how they are combined, as we will describe in this section. The basic syntactic categories are N (noun), NP (noun phrase), S (sentence), PP (prepositional phrase), AP (adverbial phrase), ADJ (adjective) and C (a special category for coordinators).

Type Raising To compactly model syntactic variations, we follow Carpenter (1997), who argues for polymorphic typing. We include the more simple, or lower type, entry in the lexicon and introduce type-raising rules to reconstruct the other when necessary at parse time. We use four rules:

$PP : g \rightarrow N\backslash N : \lambda f.\lambda x.f(x) \wedge g(x)$
$ADJ : g \rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)$
$AP : g \rightarrow S\backslash S : \lambda f.\lambda a.f(a) \wedge g(a)$
$AP : g \rightarrow S/S : \lambda f.\lambda a.f(a) \wedge g(a)$

where the first three are for prepositional, adjectival and adverbial modifications, and the fourth models the fact that adverbials are often topicalized.[4] Figures 3 and 4 show parses that use type-raising rules.

Indefinites As discussed in Section 6.1, we use a new syntactic analysis for indefinites, following Steedman (2011). Previous approaches would build parses such as

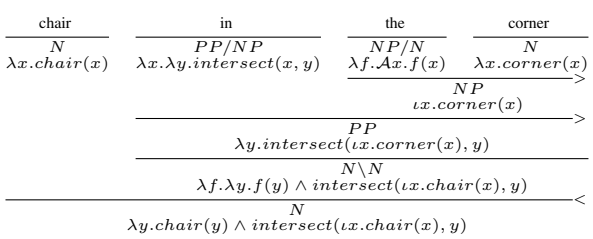
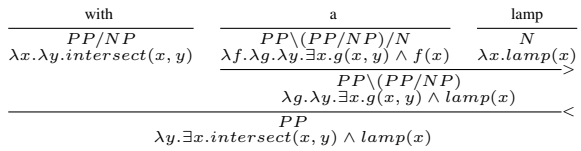


Figure 3: A CCG parse with a prepositional phrase.



where "a" has the relatively complex syntactic category $PP\backslash(PP/NP)/N$ and where similar entries would be needed to quantify over different types of verbs (e.g., $S\backslash(S/NP)/N$) and adverbials (e.g., $AP\backslash(AP/NP)/N$). Instead, we include a single lexical entry $a \vdash NP/N : \lambda f.\mathcal{A}x.f(x)$ which can be used to construct the correct meaning in all cases.

7 Joint Parsing and Execution

Our inference includes an execution component and a parser. The parser maps sentences to logical forms, and incorporates the grounded execution model. We first discuss how to execute logical forms, and then describe the joint model for execution and parsing.

7.1 Executing Logical Expressions

Dynamic Models In spatial environments, such as the ones in our task, the agent's ability to observe the world depends on its current state. Taking this aspect of spatial environments into account is challenging, but crucial for correct evaluation.

To represent the agent's point of view, for each state $s \in \mathcal{S}$, as defined in Section 2, let M_s be the state-dependent logical model. A model M consists of a domain $D_{M,T}$ of objects for each type T and an interpretation function $\mathcal{I}_{M,T} : O_T \rightarrow D_{M,T}$, where O_T is the set of T-type constants. $\mathcal{I}_{M,T}$ maps logical symbols to T-type objects, for example, it will map you to the agent's position. We have domains for position sets, actions and so on. Finally, let V_T be the set of variables of type T, and

[4] Using type-raising rules can be particularly useful when learning from sparse data. For example, it will no longer be necessary to learn three lexical entries for each adverbial phrase (with syntax AP, $S\backslash S$, and S/S).

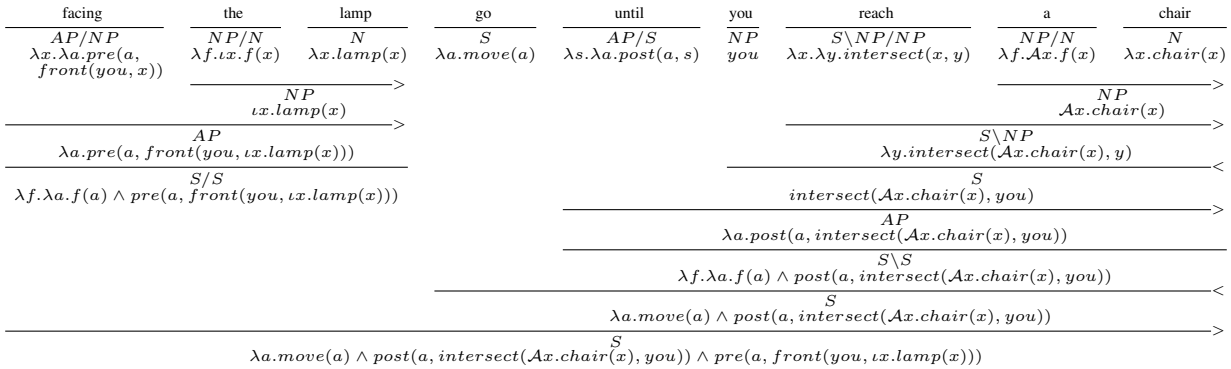


Figure 4: A CCG parse showing adverbial phrases and topicalization.

$A_T : V_T \rightarrow \bigcup_{s \in \mathcal{S}} D_{Ms,T}$ be the assignment function, which maps variables to domain objects.

For each model M_s the domain $D_{M_s,ev}$ is a set of action sequences $\{\langle a_1, ..., a_n \rangle : n \geq 1\}$. Each $\vec{a}$ defines a sequences of states s_i, as defined in Section 6.2, and associated models M_{s_i}. The key challenge for execution is that modifiers of the event will need to be evaluated under different models from this sequence. For example, consider the sentence in Figure 4. To correctly execute, the pre literal, introduced by the "facing" phrase, it must be evaluated in the model M_{s_0} for the initial state s_0. Similarly, the literal including $post$ requires the final model $M_{s_{n+1}}$. Such state dependent predicates, including pre and $post$, are called *stateful*. The list of stateful predicates is pre-defined and includes event modifiers, as well the ι quantifier, which is evaluated under M_{s_0}, since definite determiners are assumed to name objects visible from the start position. In general, a logical expression is traversed depth first and the model is updated every time a stateful predicate is reached. For example, the two e-type you constants in Figure 4 will be evaluated under different models: the one within the pre literal under M_{s_0}, and the one inside the $post$ literal under $M_{s_{n+1}}$.

Evaluation Given a logical expression l, we can compute the interpretation $\mathcal{I}_{M_{s_0},T}(l)$ by recursively mapping each subexpression to an entry on the appropriate model M.

To reflect the changing state of the agent during evaluation, we define the function $update(\vec{a}, pred)$. Given an action sequence $\vec{a}$ and a stateful predicate $pred$, $update$ returns a model M_s, where s is the state under which the literal containing $pred$ should be interpreted, either the initial state or one visited while executing $\vec{a}$. For example, given the predicate $post$ and the action sequence $\langle a_1, \ldots, a_n \rangle$, $update(\langle a_1, \ldots, a_n \rangle, post) = M_{s_{n+1}}$, where s_{n+1} the state of the agent following action a_n. By convention, we place the event variable as the first argument in literals that include one.

Given a T-type logical expression l and a starting state s_0, we compute its interpretation $\mathcal{I}_{M_{s_0},T}(l)$ recursively, following these three base cases:

- If l is a λ operator of type $\langle T_1, T_2 \rangle$ binding variable v and body b, $\mathcal{I}_{M_s,T}(l)$ is a set of pairs from $D_{T_1} \times D_{T_2}$, where $D_{T_1}, D_{T_2} \in M_s$. For each object $o \in D_{T_1}$, we create a pair (o, i) where i is the interpretation $\mathcal{I}_{M_s,T_2}(b)$ computed under a variable assignment function extended to map $A_{T_2}(v) = o$.
- If l is a literal $c(c_1, \ldots, c_n)$ with n arguments where c has type P and each c_i has type P_i, $\mathcal{I}_{M_s,T}(l)$ is computed by first interpreting the predicate c to the function $f = \mathcal{I}_{M_s,T}(c)$. In most cases, $\mathcal{I}_{M_s,T}(l) = f(\mathcal{I}_{M_s,P_1}(c_1), \ldots, \mathcal{I}_{M_s,P_n}(c_n))$. However, if c is a stateful predicate, such as pre or $post$, we instead first retrieve the appropriate new model $M_{s'} = update(\mathcal{I}_{M_s,P_1}(c_1), c)$, where c_1 is the event argument and $\mathcal{I}_{M_s,P_1}(c_1)$ is its interpretation. Then, the final results is $\mathcal{I}_{M_s,T}(l) = f(\mathcal{I}_{M_{s'},P_1}(c_1), \ldots, \mathcal{I}_{M_{s'},P_n}(c_n))$.
- If l is a T-type constant or variable, $\mathcal{I}_{M_s,T}(l)$.

The worst case complexity of the process is exponential in the number of bound variables. Although in practice we observed tractable evaluation in the majority of development cases we considered, a more comprehensive and tractable evaluation procedure is an issue that we leave for future work.

Implicit Actions Instructional language rarely specifies every action required for execution, see MacMahon (2007) for a detailed discussion in the maps domain. For example, the sentence in Figure 4 can be said even if the agent is not facing a blue hallway, with the clear implicit request that it should turn to face such a hallway before moving. To allow our agent to perform implicit actions, we extend the domain of ev-type variables by allowing the agent to prefix up to k_I action sequences before each explicit event. For example, in the agent's position in Figure 2 (set C), the set of possible events includes $\langle \text{MOVE}^I, \text{MOVE}^I, \text{RIGHT}^I, \text{MOVE} \rangle$, which contains two implicit sequences (marked by I).

Resolving Action Ambiguity Logical forms often fail to determine a unique action sequences, due to instruction ambiguity. For example, consider the instruction "go forward" and the agent state as specified in Figure 2 (set C). The instruction, which maps to $\lambda a.move(a) \wedge forward(a)$, evaluates to the set containing $\langle \text{MOVE} \rangle$, $\langle \text{MOVE}, \text{MOVE} \rangle$ and $\langle \text{MOVE}, \text{MOVE}, \text{MOVE} \rangle$, as well as five other sequences that have implicit prefixes followed by explicit MOVE actions. To resolve such ambiguity, we prefer shorter actions without implicit actions. In the example above, we will select $\langle \text{MOVE} \rangle$, which includes a single action and no implicit actions.

7.2 Joint Inference

We incorporate the execution procedure described above with a linear weighted CCG parser, as described in Section 4, to create a joint model of parsing and execution. Specifically, we execute logical forms in the current state and observe the result of their execution. For example, the word "chair" can be used to refer to different types of objects, including chairs, sofas, and barstools, in the maps domains. Our CCG grammar would include a lexical item for each meaning, but execution might fail depending on the presence of objects in the world, influencing the final parse output. Similarly, allowing implicit actions provides robustness when resolving these and other ambiguities. For example, an instruction with the precondition phrase "from the chair" might require additional actions to reach the position with the named object.

To allow such joint reasoning we define an execution e to include a parse tree e^y and trace $e^{\vec{a}}$, and define our feature function to be $\Phi(x_i, s_i, e)$, where x_i is an instruction and s_i is the start state. This approach allows joint dependencies: the state of the world influences how the agent interprets words, phrases and even complete sentences, while language understanding determines actions.

Finally, to execute sequences of instructions, we execute each starting from the end state of the previous one, using a beam of size k_s.

8 Learning

Figure 5 presents the complete learning algorithm. Our approach is online, considering each example in turn and performing two steps: expanding the lexicon and updating parameters. The algorithm assumes access to a training set $\{(x_i, s_i, \mathcal{V}_i) : i = 1 \dots n\}$, where each example includes an instruction x_i, starting state s_i and a validation function $\mathcal{V}_i$, as defined in Section 2. In addition the algorithm takes a seed lexicon Λ_0. The output is a joint model, that includes a lexicon Λ and parameters θ.

Coarse Lexical Generation To generate potential lexical entries we use the function $GENLEX(x, s, \mathcal{V}; \Lambda, \theta)$, where x is an instruction, s is a state and $\mathcal{V}$ is a validation function. Λ is the current lexicon and θ is a parameter vector. In $GENLEX$ we use coarse logical constants, as described below, to efficiently prune the set of potential lexical entries. This set is then pruned further using more precise inference in Step 1.

To compute $GENLEX$, we initially generate a large set of lexical entries and then prune most of them. The full set is generated by taking the cross product of a set of templates, computed by factoring out all templates in the seed lexicon Λ_0, and all logical constants. For example, if Λ_0 has a lexical item with the category $AP/NP : \lambda x.\lambda a.to(a, x)$ we would create entries $w \vdash AP/NP : \lambda x.\lambda a.p(a, x)$ for every phrase w in x and all constants p with the same type as to.[5]

In our development work, this approach often generated nearly 100k entries per sentence. To ease

[5] Generalizing previous work (Kwiatkowski et al., 2011), we allow templates that abstract subsets of the constants in a lexical item. For example, the seed entry $facing \vdash AP/NP : \lambda x.\lambda a.pre(a, front(you, x))$ would create 7 templates.

Inputs: Training set $\{(x_i, s_i, \mathcal{V}_i) : i = 1 \dots n\}$ where x_i is a sentence, s_i is a state and $\mathcal{V}_i$ is a validation function, as described in Section 2. Initial lexicon Λ_0. Number of iterations T. Margin γ. Beam size k for lexicon generation.

Definitions: Let an execution e include a parse tree e^y and a trace $e^{\vec{a}}$. GEN$(x, s; \Lambda)$ is the set of all possible executions for the instruction x and state s, given the lexicon Λ. $LEX(y)$ is the set of lexical entries used in the parse tree y. Let $\Phi_i(e)$ be shorthand for the feature function $\Phi(x_i, s_i, e)$ defined in Section 7.2. Define $\Delta_i(e, e') = |\Phi_i(e) - \Phi_i(e')|_1$. $GENLEX(x, s, \mathcal{V}; \lambda, \theta)$ takes as input an instruction x, state s, validation function $\mathcal{V}$, lexicon λ and model parameters θ, and returns a set of lexical entries, as defined in Section 8. Finally, for a set of executions E let $MAXV_i(E; \theta)$ be $\{e | \forall e' \in E, \langle \theta, \Phi_i(e') \rangle \leq \langle \theta, \Phi_i(e) \rangle \wedge \mathcal{V}_i(e^{\vec{a}}) = 1\}$, the set of highest scoring valid executions.

Algorithm:

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \dots T, i = 1 \dots n$:

Step 1: (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, s_i, \mathcal{V}_i; \Lambda, \theta)$, $\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let E be the k highest scoring executions from $GEN(x_i, s_i; \lambda)$ which use at most one entry from λ_G

c. Select lexical entries from the highest scoring valid parses: $\lambda_i \leftarrow \bigcup_{e \in MAXV_i(E;\theta)} LEX(e^y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

Step 2: (Update parameters)

a. Set $G_i \leftarrow MAXV_i(GEN(x_i, s_i; \Lambda); \theta)$ and $B_i \leftarrow \{e | e \in GEN(x_i, s_i; \Lambda) \wedge \mathcal{V}_i(e^{\vec{a}}) \neq 1\}$

b. Construct sets of margin violating good and bad parses:
$R_i \leftarrow \{g | g \in G_i \wedge \exists b \in B_i \; s.t. \; \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$
$E_i \leftarrow \{b | b \in B_i \wedge \exists g \in G_i \; s.t. \; \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$

c. Apply the additive update:
$\theta \leftarrow \theta + \frac{1}{|R_i|} \sum_{r \in R_i} \Phi_i(r) - \frac{1}{|E_i|} \sum_{e \in E_i} \Phi_i(e)$

Output: Parameters θ and lexicon Λ

Figure 5: The learning algorithm.

the cost of parsing at this scale, we developed a coarse-to-fine two-pass parsing approach that limits the number of new entries considered. The algorithm first parses with coarse lexical entries that abstract the identities of the logical constants in their logical forms, thereby greatly reducing the search space. It then uses the highest scoring coarse parses to constrain the lexical entries for a final, fine parse.

Formally, we construct the coarse lexicon λ_a by replacing all constants of the same type with a single newly created, temporary constant. We then parse to create a set of trees A, such that each $y \in A$

1. is a parse for sentence x, given the world state s with the combined lexicon $\Lambda \cup \lambda_a$,
2. scored higher than e^y by at least a margin of δ_L, where e^y is the tree of e, the highest scoring execution of x, at position s under the current model, s.t. $\mathcal{V}(e^{\vec{a}}) = 1$,
3. contains at most one entry from λ_a.

Finally, from each entry $l \in \{l | l \in \lambda_a \wedge l \in y \wedge y \in A\}$, we create multiple lexical entries by replacing all temporary constants with all possible appropriately typed constants from the original set. $GENLEX$ returns all these lexical entries, which will be used to form our final fine-level analysis.

Step 1: Lexical Induction To expand our model's lexicon, we use $GENLEX$ to generate candidate lexical entries and then further refine this set by parsing with the current model. Step 1(a) in Figure 5 uses $GENLEX$ to create a temporary set of potential lexical entries λ_G. Steps (b-d) select a small subset of these lexical entries to add to the current lexicon Λ: we find the k-best executions under the model, which use at most one entry from λ_G, find the entries used in the best valid executions and add them to the current lexicon.

Step 2: Parameter Update We use a variant of a loss-driven perceptron (Singh-Miller & Collins, 2007; Artzi & Zettlemoyer, 2011) for parameter updates. However, instead of taking advantage of a loss function we use a validation signal. In step (a) we collect the highest scoring valid parses and all invalid parses. Then, in step (b) we construct the set R_i of valid analyses and E_i of invalid ones, such that their model scores are not separated by a margin δ scaled by the number of wrong features (Taskar et al., 2003). Finally, step (f) applies the update.

Discussion The algorithm uses the validation signal to drive both lexical induction and parameter updates. Unlike previous work (Zettlemoyer & Collins, 2005, 2007; Artzi & Zettlemoyer, 2011), we have no access to a set of logical constants, either through the the labeled logical form or the weak supervision signal, to guide the $GENLEX$ procedure. Therefore, to avoid over-generating lexical entries, thereby making parsing and learning intractable, we leverage typing for coarse parsing to prune the generated set. By allowing a single

	Oracle	SAIL
# of instruction sequences	501	706
# of instruction sequences with implicit actions	431	
Total # of sentences	2679	3233
Avg. sentences per sequence	5.35	4.61
Avg. tokens per sentence	7.5	7.94
Vocabulary size	373	522

Table 1: Corpora statistics (lower-cased data).

new entry per parse, we create a conservative, cascading effect, whereas a lexical entry that is introduced opens the way for many other sentence to be parsed and introduce new lexical entries. Furthermore, grounded features improve parse selection, thereby generating higher quality lexical entries.

9 Experimental Setup

Data For evaluation, we use the navigation task from MacMahon et al. (2006), which includes three environments and the SAIL corpus of instructions and follower traces. Chen and Mooney (2011) segmented the data, aligned traces to instructions, and merged traces created by different subjects. The corpus includes raw sentences, without any form of linguistic annotation. The original collection process (MacMahon et al., 2006) created many uninterpretable instructions and incorrect traces. To focus on the learning and interpretation tasks, we also created a new dataset that includes only accurate instructions labeled with a single, correct execution trace. From this *oracle* corpus, we randomly sampled 164 instruction sequences (816 sentences) for evaluation, leaving 337 (1863 sentences) for training. This simple effort will allow us to measure the effects of noise on the learning approach and provides a resource for building more accurate algorithms. Table 1 compares the two sets.

Features and Parser Following Zettlemoyer and Collins (2005), we use a CKY parser with a beam of k. To boost recall, we adopt a two-pass strategy, which allows for word skipping if the initial parse fails. We use features that indicate usage of lexical entries, templates, lexemes and type-raising rules, as described in Section 6.3, and repetitions in logical coordinations. Finally, during joint parsing, we consider only parses executable at s_i as complete.

Seed Lexicon To construct our seed lexicon we labeled 12 instruction sequences with 141 lexical entries. The sequences were randomly selected from the training set, so as to include two sequences for each participant in the original experiment. Figures 3 and 4 include a sample of our seed lexicon.

	Single Sentence	Sequence
Final state validation		
Complete system	81.98 (2.33)	59.32 (6.66)
No implicit actions	77.7 (3.7)	38.46 (1.12)
No joint execution	73.27 (3.98)	31.51 (6.66)
Trace validation		
Complete system	82.74 (2.53)	58.95 (6.88)
No implicit actions	77.64 (3.46)	38.34 (6.23)
No joint execution	72.85 (4.73)	30.89 (6.08)

Table 2: Cross-validation development accuracy and standard deviation on the oracle corpus.

Initialization and Parameters We set the weight of each template indicator feature to the number of times it is used in the seed lexicon and each repetition feature to -10. Learning parameters were tuned using cross-validation on the training set: the margin δ is set to 1, the $GENLEX$ margin δ_L is set to 2, we use 6 iterations (8 for experiments on SAIL) and take the 250 top parses during lexical generation (step 1, Figure 5). For parameter update (step 2, Figure 5) we use a parser with a beam of 100. $GENLEX$ generates lexical entries for token sequences up to length 4. k_s, the instruction sequence execution beam, is set to 10. Finally, k_I is set to 2, allowing up to two implicit action sequences per explicit one.

Evaluation Metrics To evaluate single instructions x, we compare the agent's end state to a labeled state s', as described in Section 2. We use a similar method to evaluate the execution of instruction sequences $\vec{x}$, but disregard the orientation, since end goals in MacMahon et al. (2006) are defined without orientation. When evaluating logical forms we measure exact match accuracy.

10 Results

We repeated each experiment five times, shuffling the training set between runs. For the development cross-validation runs, we also shuffled the folds. As our learning approach is online, this allows us to account for performance variations arising from training set ordering. We report mean accuracy and standard deviation across all runs (and all folds).

	Single Sentence	Sequence
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
Trace validation	**65.28** (5.09)	**31.93** (3.26)
Final state validation	**64.25** (5.12)	**30.9** (2.16)

Table 3: Cross-validation accuracy and standard deviation for the SAIL corpus.

Table 2 shows accuracy for 5-fold cross-validation on the oracle training data. We first varied the validation signal by providing the complete action sequence or the final state only, as described in Section 2. Although the final state signal is weaker, the results are similar. The relatively large difference between single sentence and sequence performance is due to (1) cascading errors in the more difficult task of sequential execution, and (2) corpus repetitions, where simple sentences are common (e.g., "turn left"). Next, we disabled the system's ability to introduce implicit actions, which was especially harmful to the full sequence performance. Finally, ablating the joint execution decreases performance, showing the benefit of the joint model.

Table 3 lists cross validation results on the SAIL corpus. To compare to previous work (Chen & Mooney, 2011), we report cross-validation results over the three maps. The approach was able to correctly execute 60% more sequences then the previous state of the art (Kim & Mooney, 2012). We also outperform the results of Chen (2012), which used 30% more training data.[6] Using the weaker validation signal creates a marginal decrease in performance. However, we still outperform all previous work, despite using weaker supervision. Interestingly, these increases were achieved with a relatively simple executor, while previous work used MARCO (MacMahon et al., 2006), which supports sophisticated recovery strategies.

Finally, we evaluate our approach on the held out test set for the oracle corpus (Table 4). In contrast to experiments on the Chen and Mooney (2011) corpus, we use a held out set for evaluation. Due to this discrepancy, all development was done on the training set only. The increase in accuracy over learning with the original corpus demonstrates the significant impact of noise on our performance. In addition to execution results, we also report exact match logical form (LF) accuracy results. For this purpose, we annotated 18 instruction sequences (105 sentences) with logical forms. The gap between execution and LF accuracy can be attributed to the complexity of the linguistic representation and redundancy in instructions. These results provide a new baseline for studying learning from cleaner supervision.

[6]This additional training data isn't publicly available.

Validation	Single Sentence	Sequence	LF
Final state	77.6 (1.14)	54.63 (3.5)	44 (6.12)
Trace	78.63 (0.84)	58.05 (3.12)	51.05 (1.14)

Table 4: Oracle corpus test accuracy and standard deviation results.

11 Discussion

We showed how to do grounded learning of a CCG semantic parser that includes a joint model of meaning and context for executing natural language instructions. The joint nature allows situated cues to directly influence parsing and also enables algorithms that learn while executing instructions.

This style of algorithm, especially when using the weaker end state validation, is closely related to reinforcement learning approaches (Branavan et al., 2009, 2010). However, we differ on optimization and objective function, where we aim for minimal loss. We expect many RL techniques to be useful to scale to more complex environments, including sampling actions and using an exploration strategy.

We also designed a semantic representation to closely match the linguistic structure of instructional language, combining ideas from many semantic theories, including, for example, Neo-Davidsonian events (Parsons, 1990). This approach allowed us to learn a compact and executable grammar that generalized well. We expect, in future work, that such modeling can be reused for more general language.

Acknowledgments

The research was supported in part by DARPA under the DEFT program through the AFRL (FA8750-13-2-0019) and the CSSG (N11AP20020), the ARO (W911NF-12-1-0197), and the NSF (IIS-1115966). The authors thank Tom Kwiatkowski, Nicholas FitzGerald and Alan Ritter for helpful discussions, David Chen for providing the evaluation corpus, and the anonymous reviewers for helpful comments.

References

Artzi, Y., & Zettlemoyer, L. (2011). Bootstrapping Semantic Parsers from Conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Barwise, J., & Cooper, R. (1981). Generalized Quantifiers and Natural Language. *Linguistics and Philosophy*, *4*(2), 159–219.

Branavan, S., Chen, H., Zettlemoyer, L., & Barzilay, R. (2009). Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing*.

Branavan, S., Zettlemoyer, L., & Barzilay, R. (2010). Reading between the lines: learning to map high-level instructions to commands. In *Proceedings of the Conference of the Association for Computational Linguistics*.

Bugmann, G., Klein, E., Lauria, S., & Kyriacou, T. (2004). Corpus-based robotics: A route instruction example. In *Proceedings of Intelligent Autonomous Systems*.

Carpenter, B. (1997). *Type-Logical Semantics*. The MIT Press.

Chen, D. L. (2012). Fast Online Lexicon Learning for Grounded Language Acquisition. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Chen, D., Kim, J., & Mooney, R. (2010). Training a multilingual sportscaster: using perceptual context to learn language. *Journal of Artificial Intelligence Research*, *37*(1), 397–436.

Chen, D., & Mooney, R. (2011). Learning to Interpret Natural Language Navigation Instructions from Observations. In *Proceedings of the National Conference on Artificial Intelligence*.

Clark, S., & Curran, J. (2007). Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, *33*(4), 493–552.

Clarke, J., Goldwasser, D., Chang, M., & Roth, D. (2010). Driving Semantic Parsing from the World's Response. In *Proceedings of the Conference on Computational Natural Language Learning*.

Collins, M. (2004). Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In *New Developments in Parsing Technology*.

Davidson, D. (1967). The logical form of action sentences. *Essays on actions and events*, 105–148.

Di Eugenio, B., & White, M. (1992). On the Interpretation of Natural Language Instructions. In *Proceedings of the Conference of the Association of Computational Linguistics*.

Dzifcak, J., Scheutz, M., Baral, C., & Schermerhorn, P. (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 *Proceedings of the IEEE International Conference on Robotics and Automation*.

Goldwasser, D., Reichart, R., Clarke, J., & Roth, D. (2011). Confidence Driven Unsupervised Semantic Parsing. In *Proceedings of the Association of Computational Linguistics*.

Goldwasser, D., & Roth, D. (2011). Learning from Natural Instructions. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Heim, I., & Kratzer, A. (1998). *Semantics in Generative Grammar*. Blackwell Oxford.

Kate, R., & Mooney, R. (2006). Using String-Kernels for Learning Semantic Parsers. In *Proceedings of the Conference of the Association for Computational Linguistics*.

Kim, J., & Mooney, R. J. (2012). Unsupervised PCFG Induction for Grounded Language Learning with Highly Ambiguous Supervision. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Kollar, T., Tellex, S., Roy, D., & Roy, N. (2010). Toward Understanding Natural Language Directions. In *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*.

Krishnamurthy, J., & Mitchell, T. (2012). Weakly Supervised Training of Semantic Parsers. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

Kwiatkowski, T., Goldwater, S., Zettlemoyer, L., & Steedman, M. (2012). A probabilistic model of syntactic and semantic acquisition from child-directed utterances and their meanings. *Proceedings of the Conference of the European Chapter of the Association of Computational Linguistics*.

Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., & Steedman, M. (2010). Inducing probabilistic CCG grammars from logical form with higher-order

unification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., & Steedman, M. (2011). Lexical Generalization in CCG Grammar Induction for Semantic Parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the International Conference on Machine Learning*.

Lewis, D. (1970). General Semantics. *Synthese*, *22*(1), 18–67.

Liang, P., Jordan, M., & Klein, D. (2009). Learning semantic correspondences with less supervision. In *Proceedings of the Joint Conference of the Association for Computational Linguistics the International Joint Conference on Natural Language Processing*.

Liang, P., Jordan, M., & Klein, D. (2011). Learning Dependency-Based Compositional Semantics. In *Proceedings of the Conference of the Association for Computational Linguistics*.

MacMahon, M. (2007). *Following Natural Language Route Instructions*. Ph.D. thesis, University of Texas at Austin.

MacMahon, M., Stankiewics, B., & Kuipers, B. (2006). Walk the Talk: Connecting Language, Knowledge, Action in Route Instructions. In *Proceedings of the National Conference on Artificial Intelligence*.

Matuszek, C., FitzGerald, N., Zettlemoyer, L., Bo, L., & Fox, D. (2012). A Joint Model of Language and Perception for Grounded Attribute Learning. *Proceedings of the International Conference on Machine Learning*.

Matuszek, C., Fox, D., & Koscher, K. (2010). Following directions using statistical machine translation. In *Proceedings of the international conference on Human-robot interaction*.

Matuszek, C., Herbst, E., Zettlemoyer, L. S., & Fox, D. (2012). Learning to Parse Natural Language Commands to a Robot Control System. In *Proceedings of the International Symposium on Experimental Robotics*.

Montague, R. (1973). The Proper Treatment of Quantification in Ordinary English. *Approaches to natural language*, *49*, 221–242.

Muresan, S. (2011). Learning for Deep Language Understanding. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Parsons, T. (1990). *Events in the Semantics of English*. The MIT Press.

Singh-Miller, N., & Collins, M. (2007). Trigger-based language modeling using a loss-sensitive perceptron algorithm. In *IEEE International Conference on Acoustics, Speech and Signal Processing*.

Steedman, M. (1996). *Surface Structure and Interpretation*. The MIT Press.

Steedman, M. (2000). *The Syntactic Process*. The MIT Press.

Steedman, M. (2011). *Taking Scope*. The MIT Press.

Taskar, B., Guestrin, C., & Koller, D. (2003). Max-Margin Markov Networks. In *Proceedings of the Conference on Neural Information Processing Systems*.

Taskar, B., Klein, D., Collins, M., Koller, D., & Manning, C. (2004). Max-Margin Parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Tellex, S., Kollar, T., Dickerson, S., Walter, M., Banerjee, A., Teller, S., & Roy, N. (2011). Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation. In *Proceedings of the National Conference on Artificial Intelligence*.

Vogel, A., & Jurafsky, D. (2010). Learning to follow navigational directions. In *Proceedings of the Conference of the Association for Computational Linguistics*.

Webber, B., Badler, N., Di Eugenio, B., Geib, C., Levison, L., & Moore, M. (1995). Instructions, Intentions and Expectations. *Artificial Intelligence*, *73*(1), 253–269.

Wei, Y., Brunskill, E., Kollar, T., & Roy, N. (2009). Where To Go: Interpreting Natural Directions Using Global Inference. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Winograd, T. (1972). Understanding Natural Language. *Cognitive Psychology*, *3*(1), 1–191.

Wong, Y., & Mooney, R. (2007). Learning Synchronous Grammars for Semantic Parsing with Lambda Calculus. In *Proceedings of the Conference of the Association for Computational Linguistics*.

Zettlemoyer, L., & Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Zettlemoyer, L., & Collins, M. (2007). Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.