

Broad-coverage CCG Semantic Parsing with AMR

Yoav Artzi*

Dept. of Computer Science and Cornell Tech
Cornell University
New York, NY 10011
yoav@cs.cornell.edu

Kenton Lee Luke Zettlemoyer

Computer Science & Engineering
University of Washington
Seattle, WA 98195

{kentonl, lsz}@cs.washington.edu

Abstract

We propose a grammar induction technique for AMR semantic parsing. While previous grammar induction techniques were designed to re-learn a new parser for each target application, the recently annotated AMR Bank provides a unique opportunity to induce a single model for understanding broad-coverage newswire text and support a wide range of applications. We present a new model that combines CCG parsing to recover compositional aspects of meaning and a factor graph to model non-compositional phenomena, such as anaphoric dependencies. Our approach achieves 66.2 Smatch F1 score on the AMR bank, significantly outperforming the previous state of the art.

1 Introduction

Semantic parsers map sentences to formal representations of their meaning (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Liang et al., 2011). Existing learning algorithms have primarily focused on building actionable meaning representations which can, for example, directly query a database (Liang et al., 2011; Kwiatkowski et al., 2013) or instruct a robotic agent (Chen, 2012; Artzi and Zettlemoyer, 2013b). However, due to their end-to-end nature, such models must be re-learned for each new target application and have only been used to parse restricted styles of text, such as questions and imperatives.

Recently, AMR (Banarescu et al., 2013) was proposed as a general-purpose meaning representation language for broad-coverage text, and work is ongoing to study its use for variety of applications such as machine translation (Jones et al., 2012) and summarization (Liu et al., 2015). The

AMR meaning bank provides a large new corpus that, for the first time, enables us to study the problem of grammar induction for broad-coverage semantic parsing. However, it also presents significant challenges for existing algorithms, including much longer sentences, more complex syntactic phenomena and increased use of non-compositional semantics, such as within-sentence coreference. In this paper, we introduce a new, scalable Combinatory Categorical Grammar (CCG; Steedman, 1996, 2000) induction approach that solves these challenges with a learned joint model of both compositional and non-compositional semantics, and achieves state-of-the-art performance on AMR Bank parsing.

We map sentences to AMR structures in a two-stage process (Section 5). First, we use CCG to construct lambda-calculus representations of the compositional aspects of AMR. CCG is designed to capture a wide range of linguistic phenomena, such as coordination and long-distance dependencies, and has been used extensively for semantic parsing. To use CCG for AMR parsing we define a simple encoding for AMRs in lambda calculus, for example, as seen with the logical form z and AMR a in Figure 1 for the sentence *Pyongyang officials denied their involvement*. However, using CCG to construct such logical forms requires a new mechanism for non-compositional reasoning, for example to model the long-range anaphoric dependency introduced by *their* in Figure 1.

To represent such dependencies while maintaining a relatively compact grammar, we follow Steedman’s (2011) use of generalized Skolem terms, a mechanism to allow global references in lambda calculus. We then allow the CCG derivation to mark when non-compositional reasoning is required with underspecified placeholders. For example, Figure 1 shows an underspecified logical form u that would be constructed by the grammar with the bolded placeholder ID indicating an un-

* Work done at the University of Washington.

resolved anaphoric reference. These placeholders are resolved by a factor graph model that is defined over the output logical form and models which part of it they refer to, for example to find the referent for a pronoun. Although primarily motivated by non-compositional reasoning, we also use this mechanism to underspecify certain relations during parsing, allowing for more effective search.

Following most work in semantic parsing, we consider two learning challenges: grammar induction, which assigns meaning representations to words and phrases, and parameter estimation, where we learn a model for combining these pieces to analyze full sentences. We introduce a new CCG grammar induction algorithm which incorporates ideas from previous algorithms (Zettlemoyer and Collins, 2005; Kwiatkowski et al., 2010) in a way that scales to the longer sentences and more varied syntactic constructions observed in newswire text. During lexical generation (Section 6.1), the algorithm first attempts to use a set of templates to hypothesize new lexical entries. It then attempts to combine bottom-up parsing with top-down recursive splitting to select the best entries and learn new templates for complex syntactic and semantic phenomena, which are re-used in later sentences to hypothesize new entries.

Finally, while previous algorithms (e.g., Zettlemoyer and Collins, 2005) have assumed the existence of a grammar that can parse nearly every sentence to update its parameters, this does not hold for AMR Bank. Due to sentence complexity and search errors, our model cannot produce fully correct logical forms for a significant portion of the training data. To learn from as much of the data as possible and accelerate learning, we adopt an early update strategy to generate effective updates from partially correct analyses (Section 6.2).

We evaluate performance on the publicly available AMR Bank (Banarescu et al., 2013) and demonstrate that our modeling and learning contributions are crucial for grammar induction at this scale and achieve new state-of-the-art results for AMR parsing (Section 8). In addition, we also present, for the first time, results without surface-form alignment heuristics, which demonstrates the need for future work, especially to generalize to other languages. The source code and learned models are available online.¹

¹<http://yoavartzi.com/amr>

x : Pyongyang officials denied their involvement.

a : (d /deny-01
:ARG0(p /person
:ARG0-of(h /have-org-role-91
:ARG1(c /city
: name(n /name :op1“Pyongyang”))
:ARG2(o /official)))
:ARG1(i /involve-01 :ARG1 p))

u : $\mathcal{A}_1(\lambda d.deny-01(d) \wedge$
ARG0($d, \mathcal{A}_2(\lambda p.person(p) \wedge$
REL-of($p, \mathcal{A}_3(\lambda h.have-org-role-91(h) \wedge$
ARG1($h, \mathcal{A}_4(\lambda c.city(c) \wedge$
name($c, \mathcal{A}_5(\lambda n.name(n) \wedge$
op1($n, PYONGYANG$)))))) \wedge
REL($h, \mathcal{A}_6(\lambda o.official(o)))))) \wedge$
ARG1($d, \mathcal{A}_7(\lambda i.involve-01(i) \wedge$
ARG1($i, \mathcal{R}(\mathbf{ID}))))))$

z : $\mathcal{A}_1(\lambda d.deny-01(d) \wedge$
ARG0($d, \mathcal{A}_2(\lambda p.person(p) \wedge$
ARG0-of($p, \mathcal{A}_3(\lambda h.have-org-role-91(h) \wedge$
ARG1($h, \mathcal{A}_4(\lambda c.city(c) \wedge$
name($c, \mathcal{A}_5(\lambda n.name(n) \wedge$
op1($n, PYONGYANG$)))))) \wedge
ARG2($h, \mathcal{A}_6(\lambda o.official(o)))))) \wedge$
ARG1($d, \mathcal{A}_7(\lambda i.involve-01(i) \wedge$
ARG1($i, \mathcal{R}(2))))))$

Figure 1: A sentence (x) paired with its AMR (a), underspecified logical form (u), which contains underspecified constants in bold that are mapped to AMR relations to generate the fully specified logical form (z).

2 Technical Overview

Task Let \mathcal{X} be the set of all possible sentences and A the set of all AMR structures. Given a sentence $x \in \mathcal{X}$, we aim to generate an AMR $a \in A$. We define a simple, deterministic and invertible conversion process between AMRs and lambda-calculus logical forms; roughly speaking, each AMR variable gets its own lambda term, which is scoped as low as possible, and each AMR role becomes a binary predicate applied to these variables. Figure 1 shows an example, and the full details are provided in the supplementary materials. Therefore, henceforth we discuss the task of mapping a sentence $x \in \mathcal{X}$ to a logical form $z \in \mathcal{Z}$, where \mathcal{Z} is the set of all logical forms. For example, in Figure 1, we would map the sentence x to the logical form z . We evaluate system performance using SMATCH (Cai and Knight, 2013).

Model Given a sentence x and lexicon Λ , we generate the set of possible derivations $\text{GEN}(x, \Lambda)$ using a two-stage process (Section 5). First, we use a weighted CCG to map x to an *underspecified logical form* u (Section 5.1), a logical form with placeholder constants for unresolved elements. For example, in the underspecified logical form u in Figure 1, the constants REL-of, REL and ID are placeholders. We then resolve

these placeholders by defining a factor graph to find their optimal mapping and generate the final logical form z . In the figure, REL-of is mapped to ARG0-of, REL to ARG2 and ID to 2.

Learning We assume access to a training set of N examples $\{(x_i, z_i) : i = 1 \dots N\}$, each containing a sentence x_i and a logical form z_i . Our goal is to learn a CCG, which constitutes learning the lexicon and estimating the parameters of both the grammar and the factor graph. We define a learning procedure (Section 6) that alternates between expanding the lexicon and updating the parameters. Learning new lexical entries relies on a two-pass process that combines learning the meaning of words and new syntactic structures, and supports learning with and without alignment heuristics (e.g., from Flanigan et al., 2014).

3 Related Work

The problem of learning semantic parsers has received significant attention. Algorithms have been developed for learning from different forms of supervision, including logical forms (Wong and Mooney, 2007; Muresan, 2011), question-answer pairs (Clarke et al., 2010; Liang et al., 2011; Cai and Yates, 2013; Kwiatkowski et al., 2013), sentences paired with demonstrations (Goldwasser and Roth, 2011; Chen and Mooney, 2011), conversational logs (Artzi and Zettlemoyer, 2011), distant supervision (Krishnamurthy and Mitchell, 2012, 2015; Reddy et al., 2014) and without explicit semantic supervision (Poon, 2013).

Although we are first to consider using CCG to build AMR representations, our work is closely related to existing methods for CCG semantic parsing. Previous CCG induction techniques have either used hand-engineered lexical templates (e.g., Zettlemoyer and Collins, 2005) or learned templates from the data directly (e.g., Kwiatkowski et al., 2010, 2012). Our two-pass reasoning for lexical generation combines ideas from both methods in a way that greatly improves scalability to long, newswire-style sentences. CCG has also been used for broad-coverage recovery of first-order logic representations (Bos, 2008; Lewis and Steedman, 2013). However, this work lacked corpora to evaluate the logical forms recovered.

AMR (Banarescu et al., 2013) is a general-purpose meaning representation and has been used in a number of applications (Pan et al., 2015; Liu et al., 2015). There is also work on recovering

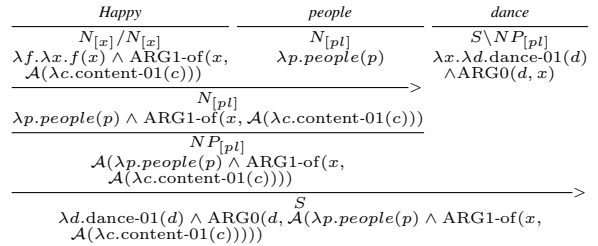


Figure 2: Example CCG tree with three lexical entries, two forward applications (>) and type-shifting of a plural noun to a noun phrase.

AMRs, including graph parsing (Flanigan et al., 2014), methods to build AMRs from dependency trees (Wang et al., 2015) and algorithms for aligning words to AMRs (Pourdamghani et al., 2014).

4 Background

Combinatory Categorical Grammar CCG is a categorial formalism that provides a transparent interface between syntax and semantics (Steedman, 1996, 2000). Section 7 details our instantiation of CCG. In CCG trees, each node is a category. Figure 2 shows a simple CCG tree. For example, $S \setminus NP_{[pl]} : \lambda x.\lambda d.\text{dance-01}(d) \wedge \text{ARG0}(d, x)$ is a category for an intransitive verb phrase. The syntactic type $S \setminus NP_{[pl]}$ indicates that an argument of type $NP_{[pl]}$ is expected and the returned syntactic type will be S . The backward slash \setminus indicates the argument is expected on the left, while a forward slash $/$ indicates it is expected on the right. The syntactic attribute pl specifies that the argument must be plural. Attribute variables enforce agreement between syntactic attributes. For example, as in Figure 2, adjectives are assigned the syntax $N_{[x]}/N_{[x]}$, where x is used to indicate that the attribute of the argument will determine the attribute of the returned category. The simply-typed lambda calculus logical form in the category represents its semantic meaning. The typing system includes basic types (e.g., entity e , truth value t) and functional types (e.g., $\langle e, t \rangle$ is the type of a function from e to t). In the example category, $\lambda x.\lambda d.\text{dance-01}(d) \wedge \text{ARG0}(d, x)$ is a $\langle e, \langle e, t \rangle \rangle$ -typed function expecting an ARG0 argument, and the conjunction specifies the roles of the dance-01 frame.

A CCG is defined by a lexicon and a set of combinators. The lexicon pairs words and phrases with their categorial meaning. For example, $\text{dance} \vdash \lambda x.\lambda d.\text{dance-01}(d) \wedge \text{ARG0}(d, x)$ pairs *dance* with the category above. We adopt a factored representation of the lexicon (Kwiatkowski et al., 2011), where entries are dynamically generated by

combining *lexemes* and *templates*. For example, the above lexical entry can be generated by pairing the lexeme $\langle \text{dance}, \{\text{dance-01}\} \rangle$ with the template $\lambda v_1.[S \setminus NP : \lambda x.\lambda a.v_1(a) \wedge \text{ARG0}(a, x)]$.

Skolem Terms and IDs Generalized Skolem terms (henceforth, *Skolem terms*) for CCG were introduced by Steedman (2011) to capture complex dependencies with relatively local quantification. We define here a simplified version of the theory to represent entities and allow distant references. Let \mathcal{A} be a $\langle \langle e, t \rangle, e \rangle$ -typed predicate. Given a $\langle e, t \rangle$ -typed logical expression C , the logical form $\mathcal{A}_n(C)$ is a Skolem term with the Skolem ID n . For example, $\mathcal{A}_2(\lambda y.\text{boy}(y))$ is a Skolem term that could represent the noun phrase *the boy*, which introduces a new entity. Skolem IDs are globally scoped, i.e., they can be referred from anywhere in the logical form without scoping constraints. To refer to Skolem terms, we define the $\langle id, e \rangle$ -typed predicate \mathcal{R} . For example, the sentence *the boy loves himself* may be represented with $\mathcal{A}_1(\lambda x.\text{love-01}(x) \wedge \text{ARG0}(x, \mathcal{A}_2(\lambda y.\text{boy}(y))) \wedge \text{ARG1}(x, \mathcal{R}(2)))$, where $\mathcal{R}(2)$ references $\mathcal{A}_2(\lambda y.\text{boy}(y))$.

5 Mapping Sentences to Logical Form

Given a sentence x and lexicon Λ , the function $\text{GEN}(x, \Lambda)$ defines the set of possible derivations. Each derivation d is a tuple $\langle y, \mathcal{M} \rangle$, where y is a CCG parse tree and \mathcal{M} is a mapping of constants from u , the *underspecified logical form* at the root of y , to their fully specified form.

5.1 Underspecified Logical Forms

An underspecified logical form represents multiple logical forms via a mapping function that maps its constants to sets of constants and Skolem IDs. For example, consider the underspecified logical form u at the top of Figure 3b. If, for example, REL can be mapped to manner or ARG2, then the sub-expression $\text{REL}(h, \mathcal{A}_6(\lambda o.\text{official}(o)))$ represents $\text{manner}(h, \mathcal{A}_6(\lambda o.\text{official}(o)))$ or $\text{ARG2}(h, \mathcal{A}_6(\lambda o.\text{official}(o)))$. During learning, we assume access to fully specified logical forms, which we convert to underspecified form as needed. In practice, all binary relations, except ARG0 and ARG1, and all Skolem ID references are underspecified.

Formally, let \mathcal{C} be the set of all constants and $\mathcal{I}(u)$ the set of all Skolem IDs in the logical form u . Let $\mathcal{S}_u : \mathcal{C} \rightarrow 2^{\mathcal{C} \cup \mathcal{I}(u)}$ be a specification func-

tion, such that its inverse is deterministic. We call a constant c a *placeholder* if $|\mathcal{S}_u(c)| > 1$. Given an underspecified logical form u , applying \mathcal{S}_u to all constants u contains, generates a set of fully specified logical forms.

We define \mathcal{S}_u to be (a) $\mathcal{S}_u(\text{ID}) = \mathcal{I}(u)$, the set of Skolem IDs in u , (b) $\mathcal{S}_u(\text{REL}) = \{\text{part}, \text{ARG2}, \dots\}$, all 67 active AMR relations except ARG0 and ARG1, (c) $\mathcal{S}_u(\text{REL-of}) = \{\text{part-of}, \text{ARG0-of}, \dots\}$, all 33 passive relations, and otherwise (d) $\mathcal{S}_u(c) = c$. For example, in u in Figure 3b, the set of assignments to the ID placeholder is $\mathcal{I}(u) = \{1, 2, 3, 4, 5, 6, 7\}$.

5.2 Derivations

The first part of a derivation $d = \langle y, \mathcal{M} \rangle$ is a CCG parse tree y with an underspecified logical form u at its root. For example, Figure 3a shows such a CCG parse tree, where the logical form contains the placeholders REL, REL-of and ID.

The second part of the derivation is a function $\mathcal{M} : \text{CONSTS}(u) \rightarrow \mathcal{C} \cup \mathcal{I}(u)$, where $\text{CONSTS}(u)$ is the set of all occurrences of constants in u . For example, in Figure 3b, $\text{CONSTS}(u)$ contains, among others, three different occurrences of ARG1 and one of ID, and \mathcal{M} maps REL to ARG2, REL-of to ARG0-of and ID to the Skolem ID 2. The set of potential assignments for each occurrence of constant c is $\mathcal{S}_u(c)$, and \mathcal{M} , which returns a single element for each constant, is a disambiguation of \mathcal{S}_u . Applying \mathcal{M} to all constants in u results in the final logical form z .

Decomposing the derivation provides two advantages. First, we are able to defer decisions from the CCG parse to the factor graph, thereby considering fewer hypotheses during parsing and simplifying the computation. Second, we can represent distant references while avoiding the complex parse trees that would have been required to represent these dependencies with scoped variables instead of Skolem IDs.²

5.3 Model

Given a sentence x , we use a weighted log-linear CCG (Lafferty et al., 2001; Clark and Curran, 2007) to rank the space of possible parses under the grammar Λ . At the root of each CCG derivation is the underspecified logical form u .

To represent a probability distribution over \mathcal{M} , we build for each u a factor graph $G_u = \langle V, F, E \rangle$,

²Similar to mention clustering methods for co-reference resolution (Ng, 2010), IDs can be viewed as creating clusters.

(a) **CCG parse y** : Maps the sentence x to an underspecified logical form u (Section 5.1) with placeholders for unresolved decisions: ID for reference identifiers and the predicates REL and REL-of for unresolved relations.

x :	<i>Pyeongyang</i>	<i>officials</i>	<i>denied</i>	<i>their</i>	<i>involvement</i>
	$NP_{[sg]}$	$N_{[pl]} \setminus (N_{[pl]}/N_{[pl]})$	$S \setminus NP/NP$	$NP_{[pl]}$	$N_{[nb]}$
	$\mathcal{A}_1(\lambda c. city(c) \wedge$ $name(c, \mathcal{A}_2(\lambda n. name(n) \wedge$ $op(n, PYONGYANG))))$	$\lambda f. \lambda p. person(p) \wedge$ $REL\text{-of}(p, \mathcal{A}_3(f(\lambda h. have\text{-org}\text{-role}\text{-91}(h) \wedge$ $REL(h, \mathcal{A}_4(\lambda o. official(o))))))$	$\lambda x. \lambda y. \lambda d. deny\text{-01}(d) \wedge$ $ARG0(d, y) \wedge$ $ARG1(d, x)$	$\mathcal{R}(\text{ID})$	$\lambda i. involve\text{-01}(i)$
u :	$\mathcal{A}_1(\lambda d. deny\text{-01}(d) \wedge ARG0(d, \mathcal{A}_2(\lambda p. person(p) \wedge REL\text{-of}(p, \mathcal{A}_3(\lambda h. have\text{-org}\text{-role}\text{-91}(h) \wedge ARG1(h, \mathcal{A}_4(\lambda c. city(c) \wedge name(c, \mathcal{A}_5(\lambda n. name(n) \wedge op(n, PYONGYANG)))))) \wedge REL(h, \mathcal{A}_6(\lambda o. official(o)))))) \wedge ARG1(d, \mathcal{A}_7(\lambda i. involve\text{-01}(i) \wedge ARG1(i, \mathcal{R}(\text{ID}))))))$				

(b) **Constant mapping \mathcal{M}** : Each constant in u , the logical form at the root of y , is mapped to a Skolem ID or a logical constant to create the fully specified logical form z , which can be converted to an AMR. Only mappings that modify constants are illustrated.

u :	$\mathcal{A}_1(\lambda d. deny\text{-01}(d) \wedge ARG0(d, \mathcal{A}_2(\lambda p. person(p) \wedge \text{REL-of}(p, \mathcal{A}_3(\lambda h. have\text{-org}\text{-role}\text{-91}(h) \wedge ARG1(h, \mathcal{A}_4(\lambda c. city(c) \wedge name(c, \mathcal{A}_5(\lambda n. name(n) \wedge op(n, PYONGYANG)))))) \wedge \text{REL}(h, \mathcal{A}_6(\lambda o. official(o)))))) \wedge ARG1(d, \mathcal{A}_7(\lambda i. involve\text{-01}(i) \wedge ARG1(i, \mathcal{R}(\text{ID}))))))$
z :	$\mathcal{A}_1(\lambda d. deny\text{-01}(d) \wedge ARG0(d, \mathcal{A}_2(\lambda p. person(p) \wedge \text{ARG0-of}(p, \mathcal{A}_3(\lambda h. have\text{-org}\text{-role}\text{-91}(h) \wedge ARG1(h, \mathcal{A}_4(\lambda c. city(c) \wedge name(c, \mathcal{A}_5(\lambda n. name(n) \wedge op(n, PYONGYANG)))))) \wedge \text{ARG2}(h, \mathcal{A}_6(\lambda o. official(o)))))) \wedge ARG1(d, \mathcal{A}_7(\lambda i. involve\text{-01}(i) \wedge ARG1(i, \mathcal{R}(2))))))$

Figure 3: A complete derivation for the sentence *Pyeongyang officials denied their involvement*.

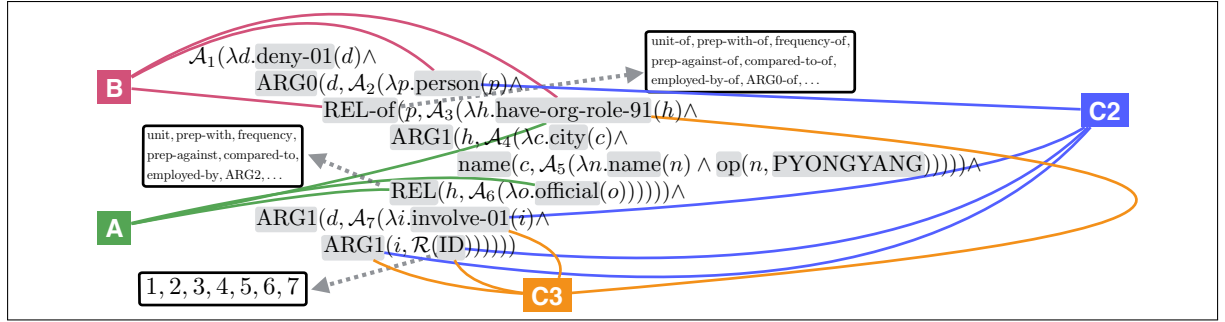


Figure 4: A visualization of the factor graph constructed for the derivation in Figure 3. Variables are marked with gray background. The set of possible assignments, marked with a dashed arrow, is only specified for placeholders (REL-of, REL and ID). Only a subset of the factors are included (A, B, C2 and C3). Solid lines represent edges. Factor A captures selectional preference between the types have-org-role-91 and official to determine the relation REL. Factor B does the same for person and have-org-role-91 to determine REF-of. Factors C2 and C3 account for selectional preferences when resolving ID. In C2, we consider the assignment 2, which will create a relation of type ARG1 between the types involve-01 and person. C3 similarly considers the assignment 3.

where $V = \text{CONSTS}(u)$ is the set of variables, F is the set of factors and E is the set of edges. Each edge is of the form (v, f) where $v \in V$ and $f \in F$. Figure 4 shows the factor graph used in generating the derivation in Figure 3, including all the variables and a subset of the factors. For each variable $v_c \in V$ such that $c \in \text{CONSTS}(u)$ the set of possible assignments is determined by $\mathcal{S}_u(c)$.

To generate the factors F and edges E we use the function $\Phi(V')$ that maps a set of variables $V' \subseteq V$ to a factor f and a set of edges, each one of the form (v, f) , where $v \in V'$. Factors express various features (Section 7), such as selectional preferences and control structures. In the figure, Factor A captures the selectional preference for the assignment of the relation REL between have-org-role-91 and official. Factor B captures a similar preference, this time to resolve

REL-of. Factor C2 captures a selectional preference triplet *involve-01/ARG1/person* that will be created if ID is resolved to the Skolem ID 2. Finally, C3 captures a similar preference for resolving ID to 3. Since the assignment of many of the variables is fixed, i.e., they are fully specified constants, in practice our factor graph representation simply conditions on them.

Derivations are scored using a log-linear model that includes both CCG parse features and those defined by the factor graph. Let $\mathcal{D}(z)$ be the subset of derivations with the final logical form z and $\theta \in \mathbb{R}^l$ be a l -dimensional parameter vector. We define the probability of the logical form z as

$$p(z|x; \theta, \Lambda) = \sum_{d \in \mathcal{D}(z)} p(d|x; \theta, \Lambda) ,$$

and the probability of a derivation d is defined as

$$p(d|x; \theta, \Lambda) = \frac{e^{\theta \cdot \phi(x, d)}}{\sum_{d' \in \text{GEN}(x, \Lambda)} e^{\theta \cdot \phi(x, d')}} , \quad (1)$$

where $\phi(x, d) \in \mathbb{R}^l$ is a feature vector (Section 7).

5.4 Inference

To compute the set of derivations $\text{GEN}(x, \Lambda)$ we define a two-stage process. We first run the CCG parser to generate underspecified logical forms. Following previous work (Zettlemoyer and Collins, 2005), we use CKY parsing to enumerate the top- K underspecified logical forms.³ During the CKY chart construction, we ignore Skolem IDs when comparing categories. This allows us to properly combine partial derivations and to fully benefit from the dynamic programming. We dynamically generate lexical entries for numbers and dates using regular expression patterns and for named-entities using a recognizer. For every underspecified logical form u , we construct a factor graph and use beam search to find the top- L configurations of the graph.⁴

During learning, we use the function $\text{GENMAX}(x, z, \theta, \Lambda)$ to get all derivations that map the sentence x to the logical form z , given parameters θ and lexicon Λ . To compute GENMAX , we follow Zettlemoyer and Collins (2005) and collect constant co-occurrence counts from z to prune from the CKY chart any category that cannot participate in a derivation leading to z . Since only constant names are changed during the second stage, setting the factor graph to get z is trivial: if the underspecified logical form is identical to z except the placeholders, we replace the placeholders with the correct final assignment, otherwise the derivation cannot result in z .

6 Learning

Learning the two-stage model requires inducing the entries of the CCG lexicon Λ and estimating the parameters θ , which score both stages of the derivation. We assume access to a training set of N examples $D = \{(x_i, z_i) : i = 1 \dots N\}$, each containing a sentence x_i and a logical form z_i . This data does not include information about the lexical entries and CCG parsing operations required to construct the correct derivations. We consider all these decisions as latent.

The main learning algorithm (Algorithm 1) starts by initializing the lexicon (line 1) and then

³See Artzi et al. (2014) for a description of this process and how to approximate the partition function in Equation 1.

⁴Experiments with loopy belief propagation showed it to be slower and less effective for our task.

Algorithm 1 The main learning algorithm.

Input: Training set $D = \{(x_i, z_i) : i = 1 \dots N\}$, number of iterations T , mini-batch size M , seed lexicon Λ_0 and learning rate μ .

Definitions: $\text{SUB}(D, i, j)$ is the set of the next j samples from D starting at i . $\text{GENMAX}(x, z, \theta, \Lambda)$ is the set of viterbi derivations from x with the final result z given parameters θ and lexicon Λ . $\text{LEX}(d)$ is the set of lexical entries used in the derivation d . $\text{COMPUTEGRAD}(x, z, \theta, \Lambda)$ computes the gradient for sentence x and logical form z , given the parameters θ and lexicon Λ , and it is described in Section 6.2. $\text{ADAGRAD}(\Delta)$ applies a per-feature learning rate to the gradient Δ (Duchi et al., 2011).

Output: Lexicon Λ and model parameters θ .

```

1:  $\Lambda \leftarrow \Lambda_0$ 
2: for  $t = 1$  to  $T$  do
3:    $\gg$  Generate entries and update the lexicon.
4:   for  $i = 1$  to  $N$  do
5:      $\lambda_{\text{new}} \leftarrow \lambda_{\text{new}} \cup \text{GENENTRIES}(x_i, z_i, \theta, \Lambda)$ 
6:    $\Lambda \leftarrow \Lambda \cup \lambda_{\text{new}}$ 
7:    $\gg$  Compute and apply mini-batch gradient updates.
8:   for  $i = 1$  to  $\lceil \frac{N}{M} \rceil$  do
9:      $\Delta \leftarrow \vec{0}$ 
10:    for  $(x, z)$  in  $\text{SUB}(D, i, M)$  do
11:       $\gg$  Compute and aggregate the gradient.
12:       $\Delta \leftarrow \Delta + \text{COMPUTEGRAD}(x, z, \theta, \Lambda)$ 
13:       $\theta \leftarrow \theta + \mu \text{ADAGRAD}(\Delta)$ 
14:       $\gg$  Get all correct viterbi derivations.
15:       $V \leftarrow \bigcup_{(x,z) \in D} \text{GENMAX}(x, z, \theta, \Lambda)$ 
16:       $\gg$  Retain only entries from derivations in  $V$ .
17:       $\Lambda \leftarrow \bigcup_{d \in V} \text{LEX}(d)$ 
18: return  $\Lambda$  and  $\theta$ 

```

Algorithm 2 GENENTRIES : Procedure to generate lexical entries from one training sample. See Section 6.1 for details.

Input: Sample (x, z) , model parameters θ and lexicon Λ .

Definitions: $\text{GENLEX}(x, z, \Lambda)$ and $\text{RECSPLIT}(z, z, \theta, \Lambda)$ are defined in Section 6.1.

Output: Set of lexical entries λ .

```

1:  $\gg$  Augment lexicon with sample-specific entries.
2:  $\Lambda_+ \leftarrow \Lambda \cup \text{GENLEX}(x, z, \Lambda)$ 
3:  $\gg$  Get max-scoring correct derivations.
4:  $\mathcal{D}_+ \leftarrow \text{GENMAX}(x, z, \Lambda_+, \theta)$ 
5: if  $|\mathcal{D}_+| > 0$  then
6:    $\gg$  Return entries from max-scoring derivations.
7:   return  $\bigcup_{d \in \mathcal{D}_+} \text{LEX}(d)$ 
8: else
9:    $\gg$  Top-down splitting to generate new entries.
10:  return  $\text{RECSPLIT}(x, z, \theta, \Lambda_+)$ 

```

processes the data T times (line 2), each time alternating between batch expansion of the lexicon and a sequence of mini-batch parameter updates. An iteration starts with a batch pass to expand the lexicon. The subroutine GENENTRIES , described in Section 6.1 and Algorithm 2, is called to generate a set of new entries for each sample (line 5).

Next, we update the parameters θ with mini-batch updates. Given a mini-batch size of M , we use the procedure $\text{SUB}(D, i, M)$ to get the i -th segment of the data D of size M . We process this segment (line 10) to accumulate the

mini-batch gradient Δ by calling the procedure $\text{COMPUTEGRAD}(x, z, \theta, \Lambda)$ (line 12), which computes the gradient for x and z given θ and Λ , as described in Section 6.2. We use AdaGrad (Duchi et al., 2011) parameter updates (line 13).

Each iteration concludes with removing all lexical entries not used in max-scoring correct derivations, to correct for overgeneration (lines 14-17).

6.1 Lexicon Expansion: GENENTRIES

Given a sentence x , a logical form z , parameters θ and a lexicon Λ , $\text{GENENTRIES}(x, z, \theta, \Lambda)$ (Algorithm 2) computes a set of lexical entries, such that there exists at least one derivation d using these entries from x to z . We first use $\text{GENLEX}(x, z, \Lambda)$ to generate a large set of potential lexical entries from u , the underspecified form of z , by generating lexemes (Section 4) and pairing them with all templates in Λ . We then use a two-pass process to select the entries to return. The set of generated lexemes is a union of: (a) the set G_{gen} that includes all pairings of subsets of constants from z with spans in x up to length k_{gen} and (b) the set that is constructed by matching named-entity constants⁵ in z with their corresponding mentions in the text to create new lexemes with potentially any other constant (for lexemes with multiple constants). Λ is augmented with the generated set of lexical entries to create Λ_+ (line 2).

First Pass Given the augmented lexicon Λ_+ , we compute the set $\mathcal{D}_+ = \text{GENMAX}(x, z, \theta, \Lambda_+)$ (line 4). Following Artzi and Zettlemoyer (2013b), we constrain the set of derivations to include only those that use at most one lexeme from G_{gen} . If generating new lexemes is sufficient to derive z from x , \mathcal{D}_+ will contain these derivations and we return their lexical entries to be added to the lexicon Λ (lines 5-7). Otherwise, we proceed to do a second pass, where we try to generate new templates to parse the sentence.

Second Pass: RECSPLIT In this pass we try to generate max-scoring derivations in a top-down process. Starting from u , the underspecified form of z , we search for CCG parsing steps that will connect to existing partial derivations in the CKY chart to create a complete parse tree. Since the space of possible operations is extremely large,

⁵Named-entity constants are created from name instances when converting from AMR to lambda calculus. See the supplementary material for the exact procedure.

we use CCGBank (Hockenmaier and Steedman, 2007) categories to prune, as described below.

The second pass is executed by calling $\text{RECSPLIT}(x, z, \theta, \Lambda_+)$, which returns a set of lexical entries to add to the model (line 10). We recursively apply the splitting operation introduced by Kwiatkowski et al. (2010). Given a CCG category, splitting outputs all possible category pairs that could have originally generated it. For example, given the category $S \setminus NP \vdash \lambda y. \lambda d. \text{deny-01}(d) \wedge \text{ARG0}(d, y) \wedge \text{ARG1}(d, \mathcal{A}_1(\lambda i. \text{involve-01}(i) \wedge \text{ARG1}(i, \mathcal{R}(\text{ID}))))$, one of the possible splits will include the categories $S \setminus NP / NP \vdash \lambda x. \lambda y. \lambda d. \text{deny-01}(d) \wedge \text{ARG0}(d, y) \wedge \text{ARG1}(d, x)$ and $NP \vdash \mathcal{A}_1(\lambda i. \text{involve-01}(i) \wedge \text{ARG1}(i, \mathcal{R}(\text{ID})))$ which would combine with forward application ($>$). Kwiatkowski et al. (2010) present the full details.⁶ The process starts from u , the underspecified form of z , and recursively applies the splitting operation while ensuring that: (1) there is at most one entry from G_{gen} or one entry where both the template and lexemes are new in the derivation, (2) each parsing step must have at least one child that may be constructed from an existing partial derivation, and (3) for each new parsing step, the syntax of a newly generated child must match the syntax of a CCGBank category for the same span. To search the space of derivations we populate a CKY chart and do a top-down beam search, where in each step we split categories for smaller spans.

6.2 Gradient Computation: COMPUTEGRAD

Given a sentence x , its labeled logical form z , parameters θ and lexicon Λ , the procedure $\text{COMPUTEGRAD}(x, z, \theta, \Lambda)$ computes the gradient for the sample (x, z) . Let $\mathcal{D}^*(z) = \text{GENMAX}(x, z, \theta, \Lambda)$, the set of max-scoring correct derivations. The *hard* gradient update is:

$$\frac{1}{|\mathcal{D}^*(z)|} \sum_{d \in \mathcal{D}^*(z)} \phi(x_i, d) - E_{p(d, |x_i; \theta, \Lambda)}[\phi(x_i, d)] \quad (2)$$

where $\phi(x, d) \in \mathbb{R}^l$ is a l -dimensional feature vector (Section 5.3) and the positive portion of the gradient, rather than using expected features, averages over all max-scoring correct derivations.

Early updates To generate an effective update when no correct derivation is observed, we follow Collins and Roark (2004) and do an early update if $\mathcal{D}^*(z)$ is empty or if $\text{GEN}(x, \Lambda)$, the set

⁶Unlike Kwiatkowski et al. (2010), we also introduce syntactic attributes (e.g., *pl*, *sg*) when splitting.

of derivations for x , does not contain a derivation with the correct final logical form z . Given the partial derivations, our gradient computation is identical to Equation 2. However, in contrast to Collins and Roark (2004) our data does not include gold derivations. Therefore, we attempt to identify max-scoring partial derivations that may lead to the correct derivation. We extract sub-expressions from u ,⁷ the underspecified form of z , and search the CKY chart for the top-scoring non-overlapping spans that contain categories with these logical forms. We use the partial derivations leading to these cells to compute the gradient.

The benefit of early updates is two-fold. First, as expected, it leads to higher quality updates that are focused on the errors the model makes. Second, given the complexity of the data, it allows us to have updates for many examples that would be otherwise ignored. In our experiments, we observe this behavior with nearly 40% of the training set.

7 Experimental Setup

Data, Tools and Metric For evaluation, we use AMR Bank release 1.0 (LDC2014T12). We use the proxy report portion, which includes newswire articles from the English Gigaword corpus, and follow the official split for training, development and evaluation (6603/826/823 sentences). We use EasyCCG (Lewis and Steedman, 2014) trained with the re-banked CCGBank (Hockenmaier and Steedman, 2007; Honnibal et al., 2010) to generate CCGBank categories, the Illinois Named Entity Tagger (Ratinov and Roth, 2009) for NER, Stanford CoreNLP (Manning et al., 2014) for tokenization and part-of-speech tagging and UW SPF (Artzi and Zettlemoyer, 2013a) to develop our system. We use SMATCH (Cai and Knight, 2013) to evaluate logical forms converted back to AMRs.

CCG We use three syntactic attributes: singular *sg*, mass nouns *nb* and plural *pl*. When factoring lexical entries, we avoid extracting binary relations and references, and leave them in the template. We use backward and forward binary combinators for application, composition and crossing composition. We allow non-crossing composition up to the third order. We also add rules to handle punctuation and unary rules for type-shifting non-adjectives in adjectival positions and verb phrases in adverbial positions. We allow

⁷We extract all sub-expressions of type e , $\langle e, t \rangle$, $\langle \langle e, t \rangle, \langle e, t \rangle \rangle$ or $\langle e, \langle e, t \rangle \rangle$ from u .

shifting of bare plurals, mass nouns and named entities to noun phrases. To avoid spurious ambiguity during parsing, we use normal-form constraints (Hockenmaier and Bisk, 2010). We use five basic lambda calculus types: entity e , truth value t , identifier id , quoted text txt and integer i .

Features During CCG parsing, we use indicator features for unary type shifting, crossing composition, lexemes, templates and dynamically generated lexical entries. We also use indicators for co-occurrence of part-of-speech tags and syntactic attributes, repetitions in logical conjunctions and attachments in the logical form. In the factor graph, we use indicator features for control structures, parent-relation-child selectional preferences and for mapping a relation to its final form. See the supplementary material for a detailed description.

Initialization and Parameters We created the seed lexicon from the training data by sampling and annotating 50 sentences with lexical entries, adding entries for pronouns and adding lexemes for all alignments generated by JAMR (Flanigan et al., 2014). We initialize features weights as follows: 10 for all lexeme feature for seed entries and entries generated by named-entity matching (Section 6.1), IBM Model 1 scores for all other lexemes (Kwiatkowski et al., 2011), -3 for unary type shifting and crossing composition features, 3 for features that pair singular and plural part-of-speech tags with singular and plural attributes and 0 for all other features. We set the number of iterations $T = 10$ and select the best model based on development results. We set the max number of tokens for lexical generation $k_{\text{gen}} = 2$, learning rate $\mu = 0.1$, CCG parsing beam $K = 50$, factor graph beam $L = 100$, mini batch size $M = 40$ and use a beam of 100 for GENMAX.

Two-pass Inference During testing, we perform two passes of inference for every sentence. First, we run our inference procedure (Section 5.4). If no derivations are generated, we run inference again, allowing the parser to skip words at a fixed cost and use the entries for *related* words if a word is unknown. We find related words in the lexicon using case, plurality and inflection string transformations. Finally, if necessary, we heuristically transform the logical forms at the root of the CCG parse trees to valid AMR logical forms. We set the cost of logical form transformation and word skipping to 10 and the cost of using related entries to 5.

8 Results

Table 1 shows SMATCH test results. We compare our approach to the latest, *fixed* version of JAMR (Flanigan et al., 2014) available online,⁸ the only system to report test results on the official LDC release. Our approach outperforms JAMR by 3 SMATCH F1 points, with a significant gain in recall. Given consensus inter-annotator agreement of 83 SMATCH F1 (Flanigan et al., 2014), this improvement reduces the gap between automated methods and human performance by 15%. Although not strictly comparable, Table 1 also includes results on the pre-release AMR Bank corpus, including the published JAMR results, their fixed results and the results of Wang et al. (2015).

Table 2 shows SMATCH scores for the development set, with ablations. The supplementary material includes example output derivations and qualitative comparison to JAMR outputs. We first remove underspecifying constants, which leaves the factor graph to resolve only references. While the expressivity of the model remains the same, more decisions are considered during parsing, modestly impacting performance.

We also study the different methods for lexical generation. Skipping the second recursive splitting pass in GENENTRIES creates an interesting tradeoff. As we are unable to learn templates without splitting, we induce a significantly smaller lexicon (500K vs. 1.6M entries). Although we are unable to recover many syntactic constructions, our search problem is in general much simpler. We therefore see a relatively mild drop in overall performance (1.1 F1). Removing G_{gen} during lexical generation (Section 6.1) creates a more significant drop in performance (3.4 F1), demonstrating how considering all possible lexemes allows the system to recover entries that are not covered by heuristic alignments. We are also able for the first time to report AMR parsing results without any surface-form similarity heuristics, by removing both JAMR alignments and named-entity matching lexical generation (Section 6.1). The significant drop in performance (20 points F1) demonstrates the need for better alignment algorithm.

Finally, Figure 5 plots development SMATCH F1 with and without early updates. As expected, early updates increase the learning rate significantly and have a large impact on overall performance. Without early updates we are unable to

⁸JAMR is available at <http://tiny.cc/jamr>.

	P	R	F1
JAMR (fixed)	67.8	59.2	63.2
Our approach	66.8	65.7	66.3
Pre-release corpus results			
JAMR (Flanigan et al., 2014)	52.0	66.0	58.0
JAMR (fixed)	66.8	58.3	62.3
Wang et al. (2015)	64.0	62.0	63.0

Table 1: Test SMATCH results.

	P	R	F1
Full system	67.2	65.1	66.1
w/o underspecified constants	66.9	64.2	65.5
Lexical learning ablations			
w/o splitting	65.0	65.0	65.0
w/o G_{gen}	62.6	62.7	62.6
w/o surface-form similarity	55.9	38.5	45.6

Table 2: Development SMATCH results.

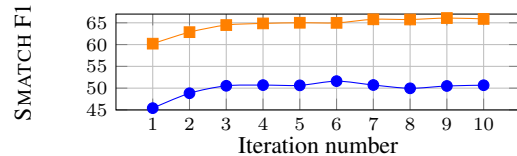


Figure 5: Development SMATCH F1 without early updates (●) and with early updates (■).

learn from almost half of the data, and performance drops by nearly 15 points.

9 Conclusion

We described an approach for broad-coverage CCG induction for semantic parsing, including a joint representation of compositional and non-compositional semantics, a new grammar induction technique and an early update procedure. We used AMR as the target representation and present new state-of-the-art AMR parsing results.

While we focused on recovering non-compositional dependencies, other non-compositional phenomena remain to be studied. Although our technique is able to learn certain idioms as multi-word phrases, learning to recognize discontinuous idioms remains open. Similarly, resolving cross-sentence references, which are not annotated in AMR Bank, is important future work. Finally, we would like to reduce the dependency on surface-form heuristics, for example to better generalize to other languages.

Acknowledgements

This research was supported in part by a Microsoft Research PhD Fellowship, NSF (IIS-1252835), DARPA under the DEFT program through AFRL (FA8750-13-2-0019), an Allen Distinguished Investigator Award and gifts from Google and Amazon. The authors thank Mark Yatskar, Tom Kwiatkowski, Chloé Kiddon, Eunsol Choi, Mike Lewis and the reviewers for their helpful advice.

References

- Artzi, Y., Das, D., and Petrov, S. (2014). Learning compact lexicons for CCG semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.
- Artzi, Y. and Zettlemoyer, L. S. (2011). Bootstrapping semantic parsers from conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.
- Artzi, Y. and Zettlemoyer, L. S. (2013a). UW SPF: The University of Washington Semantic Parsing Framework.
- Artzi, Y. and Zettlemoyer, L. S. (2013b). Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1(1):49–62.
- Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., and Schneider, N. (2013). Abstract meaning representation for sembanking. In *Proceedings of the Linguistic Annotation Workshop*.
- Bos, J. (2008). Wide-coverage semantic analysis with Boxer. In *Proceedings of the Conference on Semantics in Text Processing*.
- Cai, Q. and Yates, A. (2013). Semantic parsing Freebase: Towards open-domain semantic parsing. In *Proceedings of the Joint Conference on Lexical and Computational Semantics*.
- Cai, S. and Knight, K. (2013). Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the Conference of the Association of Computational Linguistics*.
- Chen, D. (2012). Fast online lexicon learning for grounded language acquisition. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.
- Chen, D. L. and Mooney, R. J. (2011). Learning to interpret natural language navigation instructions from observations. In *Proceedings of the National Conference on Artificial Intelligence*.
- Clark, S. and Curran, J. R. (2007). Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.
- Clarke, J., Goldwasser, D., Chang, M., and Roth, D. (2010). Driving semantic parsing from the world’s response. In *Proceedings of the Conference on Computational Natural Language Learning*.
- Collins, M. and Roark, B. (2004). Incremental parsing with the perceptron algorithm. In *Proceedings of the Annual Meeting on Association for Computational Linguistics*.
- Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, pages 2121–2159.
- Flanigan, J., Thomson, S., Carbonell, J., Dyer, C., and Smith, N. A. (2014). A discriminative graph-based parser for the Abstract Meaning Representation. In *Proceedings of the Conference of the Association of Computational Linguistics*.
- Goldwasser, D. and Roth, D. (2011). Learning from natural instructions. In *Proceedings of the International Joint Conference on Artificial Intelligence*.
- Hockenmaier, J. and Bisk, Y. (2010). Normal-form parsing for combinatory categorial grammars with generalized composition and type-raising. In *Proceedings of the International Conference on Computational Linguistics*.
- Hockenmaier, J. and Steedman, M. (2007). CCG-Bank: A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, pages 355–396.
- Honnibal, M., Curran, J. R., and Bos, J. (2010). Rebanking CCGBank for Improved NP Interpretation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.
- Jones, B., Andreas, J., Bauer, D., Hermann, K. M., and Knight, K. (2012). Semantics-based machine translation with hyperedge replacement grammars. In *Proceedings of the International Conference on Computational Linguistics*.
- Krishnamurthy, J. and 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*.

- Krishnamurthy, J. and Mitchell, T. M. (2015). Learning a compositional semantics for Freebase with an open predicate vocabulary. *Transactions of the Association for Computational Linguistics*, 3.
- Kwiatkowski, T., Choi, E., Artzi, Y., and Zettlemoyer, L. S. (2013). Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.
- Kwiatkowski, T., Goldwater, S., Zettlemoyer, L. S., and 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. S., Goldwater, S., and 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. S., Goldwater, S., and 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., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*.
- Lewis, M. and Steedman, M. (2013). Combined distributional and logical semantics. *Transactions of the Association for Computational Linguistics*, 1.
- Lewis, M. and Steedman, M. (2014). A* CCG parsing with a supertag-factored model. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.
- Liang, P., Jordan, M., and Klein, D. (2011). Learning dependency-based compositional semantics. In *Proceedings of the Conference of the Association for Computational Linguistics*.
- Liu, F., Flanigan, J., Thomson, S., Sadeh, N., and Smith, N. A. (2015). Toward abstractive summarization using semantic representations. In *Proceedings of the North American Association for Computational Linguistics*.
- Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.
- Muresan, S. (2011). Learning for deep language understanding. In *Proceedings of the International Joint Conference on Artificial Intelligence*.
- Ng, V. (2010). Supervised noun phrase coreference research: The first fifteen years. In *Proceedings of the annual meeting of the association for computational linguistics*.
- Pan, X., Cassidy, T., Hermjakob, U., Ji, H., and Knight, K. (2015). Unsupervised entity linking with Abstract Meaning Representation. In *Proceedings of the North American Association for Computational Linguistics*.
- Poon, H. (2013). Grounded unsupervised semantic parsing. In *Association for Computational Linguistics (ACL)*.
- Pourdamghani, N., Gao, Y., Hermjakob, U., and Knight, K. (2014). Aligning English strings with Abstract Meaning Representation graphs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.
- Ratinov, L. and Roth, D. (2009). Design challenges and misconceptions in named entity recognition. In *Proceedings of the Conference on Computational Natural Language Learning*.
- Reddy, S., Lapata, M., and Steedman, M. (2014). Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2.
- 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.
- Wang, C., Xue, N., Pradhan, S., and Pradhan, S. (2015). A transition-based algorithm for AMR parsing. In *Proceedings of the North American Association for Computational Linguistics*.
- Wong, Y. and Mooney, R. J. (2007). Learning synchronous grammars for semantic parsing with

lambda calculus. In *Proceedings of the Conference of the Association for Computational Linguistics*.

Zelle, J. and Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. In *Proceedings of the National*

Conference on Artificial Intelligence.

Zettlemoyer, L. S. and 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*.