Transforming Dependency Structures to Logical Forms for Semantic Parsing

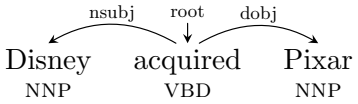
Siva Reddy[†a] **Oscar Täckström**[‡] **Michael Collins**[‡b] **Tom Kwiatkowski**[‡]
Dipanjan Das[‡] **Mark Steedman**[†] **Mirella Lapata**[†]
[†]ILCC, School of Informatics, University of Edinburgh
[‡] Google, New York
siva.reddy@ed.ac.uk
{oscart, mjcollins, tomkwiat, dipanjand}@google.com
{steedman, mlap}@inf.ed.ac.uk

Abstract

The strongly typed syntax of grammar formalisms such as CCG, TAG, LFG and HPSG offers a synchronous framework for deriving syntactic structures and semantic logical forms. In contrast—partly due to the lack of a strong type system—dependency structures are easy to annotate and have become a widely used form of syntactic analysis for many languages. However, the lack of a type system makes a formal mechanism for deriving logical forms from dependency structures challenging. We address this by introducing a robust system based on the lambda calculus for deriving neo-Davidsonian logical forms from dependency trees. These logical forms are then used for semantic parsing of natural language to Freebase. Experiments on the Free917 and WebQuestions datasets show that our representation is superior to the original dependency trees and that it outperforms a CCG-based representation on this task. Compared to prior work, we obtain the strongest result to date on Free917 and competitive results on WebQuestions.

1 Introduction

Semantic parsers map sentences onto logical forms that can be used to query databases (Zettlemoyer and Collins, 2005; Wong and Mooney, 2006), instruct robots (Chen and Mooney, 2011), extract information (Krishnamurthy and Mitchell, 2012), or describe visual scenes (Matuszek et al., 2012). Current systems accomplish this by learning task-specific grammars (Berant et al., 2013), by using strongly-typed CCG grammars (Reddy et al., 2014), or by eschewing the use of a grammar entirely (Yih et al., 2015).



(a) The dependency tree for *Disney acquired Pixar*.

(nsubj (dobj acquired Pixar) Disney)

(b) The s-expression for the dependency tree.

$$\lambda x.\, \exists yz.\, \text{acquired}(x_e) \wedge \text{Disney}(y_a) \wedge \text{Pixar}(z_a) \wedge \text{arg}_1(x_e, y_a) \wedge \text{arg}_2(x_e, z_a)$$

(c) The composed lambda-calculus expression.

Figure 1: The dependency tree is binarized into its s-expression, which is then composed into the lambda expression representing the sentence logical form.

In recent years, there have been significant advances in developing fast and accurate dependency parsers for many languages (McDonald et al., 2005; Nivre et al., 2007; Martins et al., 2013, *inter alia*). Motivated by the desire to carry these advances over to semantic parsing tasks, we present a robust method for mapping dependency trees to logical forms that represent underlying predicate-argument structures.[1] We empirically validate the utility of these logical forms for question answering from databases. Since our approach uses dependency trees as input, we hypothesize that it will generalize better to domains that are well covered by dependency parsers than methods that induce semantic grammars from scratch.

The system that maps a dependency tree to its logical form (henceforth DEPLAMBDA) is illustrated in Figure 1. First, the dependency tree is binarized via an obliqueness hierarchy to give an *s-expression* that describes the application of functions to pairs

[a] Work carried out during an internship at Google.
[b] On leave from Columbia University.

[1] By "robust", we refer to the ability to gracefully handle parse errors as well as the untyped nature of dependency syntax.

Transactions of the Association for Computational Linguistics, vol. 4, pp. 127–140, 2016. Action Editor: Christopher Potts.
Submission batch: 12/2015; Published 4/2016.

of arguments. Each node in this s-expression is then substituted for a lambda-calculus expression and the relabeled s-expression is beta-reduced to give the logical form in Figure 1(c). Since dependency syntax does not have an associated type theory, we introduce a type system that assigns a single type to all constituents, thus avoiding the need for type checking (Section 2). DEPLAMBDA uses this system to generate robust logical forms, even when the dependency structure does not mirror predicate-argument relationships in constructions such as conjunctions, prepositional phrases, relative clauses, and wh-questions (Section 3).

These *ungrounded* logical forms (Kwiatkowski et al., 2013; Reddy et al., 2014; Krishnamurthy and Mitchell, 2015) are used for question answering against Freebase, by passing them as input to GRAPHPARSER (Reddy et al., 2014), a system that learns to map logical predicates to Freebase, resulting in *grounded* Freebase queries (Section 4). We show that our approach achieves state-of-the-art performance on the Free917 dataset and competitive performance on the WebQuestions dataset, whereas building the Freebase queries directly from dependency trees gives significantly lower performance. Finally, we show that our approach outperforms a directly comparable method that generates ungrounded logical forms using CCG. Details of our experimental setup and results are presented in Section 5 and Section 6, respectively.

2 Logical Forms

We use a version of the lambda calculus with three base types: individuals ($\mathbf{Ind}$), events ($\mathbf{Event}$), and truth values ($\mathbf{Bool}$). Roughly speaking individuals are introduced by nouns, events are introduced by verbs, and whole sentences are functions onto truth values. For types $\mathbf{A}$ and $\mathbf{B}$, we use $\mathbf{A} \times \mathbf{B}$ to denote the product type, while $\mathbf{A} \to \mathbf{B}$ denotes the type of functions mapping elements of $\mathbf{A}$ to elements of $\mathbf{B}$. We will make extensive use of variables of type $\mathbf{Ind} \times \mathbf{Event}$. For any variable x of type $\mathbf{Ind} \times \mathbf{Event}$, we use $x = (x_a, x_e)$ to denote the pair of variables x_a (of type $\mathbf{Ind}$) and x_e (of type $\mathbf{Event}$). Here, the subscript denotes the projections $\cdot_a : \mathbf{Ind} \times \mathbf{Event} \to \mathbf{Ind}$ and $\cdot_e : \mathbf{Ind} \times \mathbf{Event} \to \mathbf{Event}$.

An important constraint on the lambda calculus system is as follows: *All natural language constituents have a lambda-calculus expression of type* $\mathbf{Ind} \times \mathbf{Event} \to \mathbf{Bool}$.

A "constituent" in this definition is either a single word, or an s-expression. S-expressions are defined formally in the next section; examples are $(\text{dobj acquired Pixar})$ and $(\text{nsubj } (\text{dobj acquired Pixar}) \text{ Disney})$. Essentially, s-expressions are binarized dependency trees, which include an ordering over the different dependencies to a head (in the above the *dobj* modifier is combined before the *nsubj* modifier).

Some examples of lambda-calculus expressions for single words (lexical entries) are as follows:

$$\begin{aligned} \text{acquired} &\Rightarrow \lambda x.\, \text{acquired}(x_e) \\ \text{Disney} &\Rightarrow \lambda y.\, \text{Disney}(y_a) \\ \text{Pixar} &\Rightarrow \lambda z.\, \text{Pixar}(z_a) \end{aligned}$$

An example for a full sentence is as follows:

$$\begin{aligned} &\text{Disney acquired Pixar} \Rightarrow \\ &\lambda x.\, \exists y z.\, \text{acquired}(x_e) \wedge \text{Disney}(y_a) \\ &\qquad \wedge \text{Pixar}(z_a) \wedge \text{arg}_1(x_e, y_a) \wedge \text{arg}_2(x_e, z_a) \end{aligned}$$

This is a neo-Davidsonian style of analysis. Verbs such as *acquired* make use of event variables such as x_e, whereas nouns such as *Disney* make use of individual variables such as y_a.

The restriction that all expressions are of type $\mathbf{Ind} \times \mathbf{Event} \to \mathbf{Bool}$ simplifies the type system considerably. While it leads to difficulty with some linguistic constructions—see Section 3.3 for some examples—we believe the simplicity and robustness of the resulting system outweighs these concerns. It also leads to some spurious variables that are bound by lambdas or existentials, but which do not appear as arguments of any predicate: for example in the above analysis for *Disney acquired Pixar*, the variables x_a, y_e and z_e are unused. However these "spurious" variables are easily identified and discarded.

An important motivation for having variables of type $\mathbf{Ind} \times \mathbf{Event}$ is that a single lexical item sometimes makes use of both types of variables. For example, the noun phrase *president in 2009* has semantics

$$\begin{aligned} &\lambda x.\, \exists y.\, \text{president}(x_a) \wedge \text{president\_event}(x_e) \wedge \\ &\qquad \text{arg}_1(x_e, x_a) \wedge 2009(y_a) \wedge \text{prep.in}(x_e, y_a) \end{aligned}$$

In this example *president* introduces the predicates president, corresponding to an individual, and president_event, corresponding to an event; essentially a presidency event that may have various properties. This follows the structure of Freebase closely:

Freebase contains an individual corresponding to *Barack Obama*, with a *president* property, as well as an event corresponding to the *Obama presidency*, with various properties such as a start and end date, a location, and so on. The entry for *president* is then

$$\lambda x.\, \text{president}(x_a) \wedge \text{president\_event}(x_e) \wedge \text{arg}_1(x_e, x_a)$$

Note that proper nouns do not introduce an event predicate, as can be seen from the entries for *Disney* and *Pixar* above.

3 Dependency Structures to Logical Forms

We now describe the system used to map dependency structures to logical forms. We first give an overview of the approach, then go into detail about various linguistic constructions.

3.1 An Overview of the Approach

The transformation of a dependency tree to its logical form is accomplished through a series of three steps: *binarization*, *substitution*, and *composition*. Below, we outline these steps, with some additional remarks.

Binarization. A dependency tree is mapped to an s-expression (borrowing terminology from Lisp). For example, *Disney acquired Pixar* has the s-expression

$$(\text{nsubj}\ (\text{dobj}\ \text{acquired}\ \text{Pixar})\ \text{Disney})$$

Formally, an s-expression has the form $(\text{exp1}\ \text{exp2}\ \text{exp3})$, where exp1 is a dependency label, and both exp2 and exp3 are either (1) a word such as *acquired*; or (2) an s-expression such as $(\text{dobj}\ \text{acquired}\ \text{Pixar})$.

We refer to the process of mapping a dependency tree to an s-expression as *binarization*, as it involves an ordering of modifiers to a particular head, similar to binarization of a context-free parse tree.

Substitution. Each symbol (word or label) in the s-expression is assigned a lambda expression. In our running example we have the following assignments:

acquired	$\Rightarrow \lambda x.\, \text{acquired}(x_e)$
Disney	$\Rightarrow \lambda y.\, \text{Disney}(y_a)$
Pixar	$\Rightarrow \lambda z.\, \text{Pixar}(z_a)$
nsubj	$\Rightarrow \lambda f g z.\, \exists x.\, f(z) \wedge g(x) \wedge \text{arg}_1(z_e, x_a)$
dobj	$\Rightarrow \lambda f g z.\, \exists x.\, f(z) \wedge g(x) \wedge \text{arg}_2(z_e, x_a)$

Composition. Beta-reduction is used to compose the lambda-expression terms to compute the final semantics for the input sentence. In this step expressions of the form $(\text{exp1}\ \text{exp2}\ \text{exp3})$ are interpreted as function exp1 being applied to arguments exp2 and exp3. For example, $(\text{dobj}\ \text{acquired}\ \text{Pixar})$ receives the following expression after composition:

$$\lambda z.\, \exists x.\, \text{acquired}(z_e) \wedge \text{Pixar}(x_a) \wedge \text{arg}_2(z_e, x_a)$$

Obliqueness Hierarchy. The binarization stage requires a strict ordering on the different modifiers to each head in a dependency parse. For example, in $(\text{nsubj}\ (\text{dobj}\ \text{acquired}\ \text{Pixar})\ \text{Disney})$, the dobj is attached before the nsubj. The ordering is very similar to the obliqueness hierarchy in syntactic formalisms such as HPSG (Pollard and Sag, 1994).

Type for Dependency Labels. Recall from Section 2 that every s-expression subtree receive a logical form of type $\eta = \mathbf{Ind} \times \mathbf{Event} \rightarrow \mathbf{Bool}$. It follows that in any s-expression $(\text{exp1}\ \text{exp2}\ \text{exp3})$, exp1 has type $\eta \rightarrow (\eta \rightarrow \eta)$, exp2 and exp3 both have type η, and the full expression has type η. Since each labeled dependency relation (e.g., nsubj, dobj, partmod) is associated with exp1 in connecting two s-expression subtrees, dependency labels always receive expressions of type $\eta \rightarrow (\eta \rightarrow \eta)$.

Mirroring Dependency Structure. Whenever a dependency label receives an expression of the form

$$\lambda f g z.\, \exists x.\, f(z) \wedge g(x) \wedge \text{rel}(z_e, x_a) \qquad (1)$$

where rel is a logical relation, the composition operation builds a structure that essentially mirrors the original dependency structure. For example nsubj and dobj receive expressions of this form, with rel = arg_1 and rel = arg_2, respectively; the final lambda expression for *Disney acquired Pixar* is

$$\lambda x.\, \exists y z.\, \text{acquired}(x_e) \wedge \text{Disney}(y_a) \wedge \text{Pixar}(z_a) \wedge \text{arg}_1(x_e, y_a) \wedge \text{arg}_2(x_e, z_a)$$

This structure is isomorphic to the original dependency structure: there are variables x_e, y_a and z_a corresponding to *acquired*, *Disney* and *Pixar*, respectively; and the sub-expressions $\text{arg}_1(x_e, y_a)$ and $\text{arg}_2(x_e, z_a)$ correspond to the dependencies *acquired* $\rightarrow$ *Disney* and *acquired* $\rightarrow$ *Pixar*.

By default we assume that the predicate argument structure is isomorphic to the dependency structure

and many dependency labels receive a semantics of the form shown in (1). However, there are a number of important exceptions. As one example, the dependency label $\mathrm{partmod}$ receives semantics

$$\lambda fgz.\, \exists x.\, f(z) \wedge g(x) \wedge \mathrm{arg}_1(x_e, z_a)$$

with $\mathrm{arg}_1(x_e, z_a)$ in place of the $\mathrm{arg}_1(z_e, x_a)$ in (1). This reverses the dependency direction to capture the predicate-argument structure of reduced relative constructions such as *a company acquired by Disney*.

Post-processing. We apply three post-processing steps—simple inferences over lambda-calculus expressions—to the derived logical forms. These relate to the handling of prepositions, coordination and control and are described and motivated in more detail under the respective headings below.

3.2 Analysis of Some Linguistic Constructions

In this section we describe in detail how various linguistic constructions not covered by the rule in (1)—prepositional phrases, conjunction, relative clauses, and Wh questions—are handled in the formalism.[2]

Prepositional Phrases. Prepositional phrase modifiers to nouns and verbs have similar s-expressions:

(prep president (pobj in 2009))
(prep acquired (pobj in 2009))

The following entries are used in these examples:

$\text{in} \Rightarrow \lambda x.\, \mathrm{in}(x_e)$
$\text{prep} \Rightarrow \lambda fgz.\, \exists x.\, f(z) \wedge g(x) \wedge \mathrm{prep}(z_e, x_a)$
$\text{pobj} \Rightarrow \lambda fgz.\, \exists x.\, f(z) \wedge g(x) \wedge \mathrm{pobj}(z_e, x_a)$
$\text{president} \Rightarrow \lambda x.\, \mathrm{president}(x_a)$
$\qquad \wedge \mathrm{president\_event}(x_e) \wedge \mathrm{arg}_1(x_e, x_a)$
$\text{acquired} \Rightarrow \lambda x.\, \mathrm{acquired}(x_e)$

where the entries for prep and pobj simply mirror the original dependency structure with prep modifying the event variable z_e.

The semantics for *acquired in 2009* is as follows:

$$\lambda x.\, \exists py.\, \mathrm{acquired}(x_e) \wedge 2009(y_a) \wedge \mathrm{in}(p_e) \wedge \mathrm{prep}(x_e, p_e) \wedge \mathrm{pobj}(p_e, y_a)$$

We replace $\mathrm{in}(p_e) \wedge \mathrm{prep}(x_e, p_e) \wedge \mathrm{pobj}(p_e, y_a)$ by $\mathrm{prep.in}(x_e, y_a)$ as a post-processing step, effectively collapsing out the p variable while replacing the prep and pobj dependencies by a single dependency, $\mathrm{prep.in}$. The final semantics are then as follows:

$$\lambda x.\, \exists y.\, \mathrm{acquired}(x_e) \wedge 2009(y_a) \wedge \mathrm{prep.in}(x_e, y_a)$$

In practice this step is easily achieved by identifying variables (in this case p_e) participating in prep and pobj relations. It would be tempting to achieve this step within the lambda calculus expressions themselves, but we have found the post-processing step to be more robust to parsing errors and corner cases in the usage of the prep and pobj dependency labels.

Conjunctions. First consider a simple case of NP-conjunction, *Bill and Dave founded HP*, whose s-expression is as follows:

(nsubj (dobj founded HP)
(conj-np (cc Bill and) Dave))

We make use of the following entries:

$\text{conj-np} \Rightarrow \lambda fgx.\, \exists yz.\, f(y) \wedge g(z) \wedge \mathrm{coord}(x, y, z)$
$\text{cc} \Rightarrow \lambda fgz.\, f(z)$

The sentence *Bill and Dave founded HP* then receives the following semantics:

$$\lambda e.\, \exists xyzu.\, \mathrm{Bill}(y_a) \wedge \mathrm{Dave}(z_a) \wedge \mathrm{founded}(e_e) \wedge \mathrm{HP}(u_a) \wedge \mathrm{coord}(x, y, z) \wedge \mathrm{arg}_1(e_e, x_a) \wedge \mathrm{arg}_2(e_e, u_a)$$

Note how the x variable occurs in two subexpressions: $\mathrm{coord}(x, y, z)$, and $\mathrm{arg}_1(e_e, x_a)$. It can be interpreted as a variable that conjoins variables y and z together. In particular, we introduce a post-processing step where the subexpression $\mathrm{coord}(x, y, z) \wedge \mathrm{arg}_1(e_e, x_a)$ is replaced with $\mathrm{arg}_1(e_e, y_a) \wedge \mathrm{arg}_1(e_e, z_a)$, and the x variable is removed. The resulting expression is as follows:

$$\lambda e.\, \exists yzu.\, \mathrm{Bill}(y_a) \wedge \mathrm{Dave}(z_a) \wedge \mathrm{founded}(e_e) \wedge \mathrm{HP}(u_a) \wedge \mathrm{arg}_1(e_e, y_a) \wedge \mathrm{arg}_1(e_e, z_a) \wedge \mathrm{arg}_2(e_e, u_a)$$

VP-coordination is treated in a very similar way. Consider the sentence *Eminem signed to Interscope and discovered 50 Cent*. This has the following s-expression:

(nsubj (conj-vp (cc s-to-I and) d-50) Eminem)

where s-to-I refers to the VP *signed to Interscope*, and d-50 refers to the VP *discovered 50 Cent*. The lambda-calculus expression for conj-vp is identical to the expression for conj-np:

[2] The system contains 32 binarization rules (e.g., rules for obliqueness hierarchy and identifying traces) and 46 substitution rules (i.e., rules for dependency labels and parts of speech). The rules can be found at http://github.com/sivareddyg/deplambda.

conj-vp $\Rightarrow \lambda fgx.\, \exists yz.\, f(y) \wedge g(z) \wedge \text{coord}(x, y, z)$

The logical form for the full sentence is then

$$\lambda e.\, \exists xyz.\, \text{Eminem}(x_a) \wedge \text{coord}(e, y, z) \\ \wedge \arg_1(e_e, x_a) \wedge \text{s\_to\_I}(y) \wedge \text{d\_50}(z)$$

where we use $\text{s\_to\_I}(y)$ and $\text{d\_50}(z)$ as shorthand for the lambda-calculus expressions for the two VPs.

After post-processing this is simplified to

$$\lambda e.\, \exists xyz.\, \text{Eminem}(x_a) \wedge \arg_1(y_e, x_a) \\ \wedge \arg_1(z_e, x_a) \wedge \text{s\_to\_I}(y) \wedge \text{d\_50}(z)$$

Other types of coordination, such as sentence-level coordination and PP coordination, are handled with the same mechanism. All coordination dependency labels have the same semantics as conj-np and conj-vp. The only reason for having distinct dependency labels for different types of coordination is that different labels appear in different positions in the obliqueness hierarchy. This is important for getting the correct scope for different forms of conjunction. For instance, the following s-expression for the *Eminem* example would lead to an incorrect semantics:

(conj-vp (nsubj (cc s-to-I and) Eminem) d-50)

This s-expression is not possible under the obliqueness hierarchy, which places nsubj modifiers to a verb after conj-vp modifiers.

We realize that this treatment of conjunction is quite naive in comparison to that on offer in CCG. However, given the crude analysis of conjunction in dependency syntax, a more refined treatment is beyond the scope of the current approach.

Relative Clauses. Our treatment of relative clauses is closely related to the mechanism for traces described by Moortgat (1988; 1991); see also Carpenter (1998) and Pereira (1990). Consider the NP *Apple which Jobs founded* with s-expression:

(rcmod Apple
(wh-dobj (BIND f (nsubj (dobj founded f) Jobs))
which))

Note that the s-expression has been augmented to include a variable f in dobj position, with *(*BIND f *...)* binding this variable at the clause level. These annotations are added using a set of heuristic rules over the original dependency parse tree.

The BIND operation is interpreted in the following way. If we have an expression of the form

$$(\text{BIND}\ \ f\ \ \lambda x.\, g(x))$$

where f is a variable and g is an expression that includes f, this is converted to

$$\lambda z.\, \exists x.\, g(x)\ |_{f=\text{EQ}(z)}$$

where $g(x)\ |_{f=\text{EQ}(z)}$ is the expression $g(x)$ with the expression $\text{EQ}(z)$ substituted for f. $\text{EQ}(z)(z')$ is true iff z and z' are equal (refer to the same entity). In addition we assume the following entries:

wh-dobj $\Rightarrow \lambda fgz.\, f(z)$
rcmod $\Rightarrow \lambda fgz.\, f(z) \wedge g(z)$

It can be verified that *(*BIND f (nsubj (dobj founded f*)* Jobs*))* has semantics

$$\lambda u.\, \exists xyz.\, \text{founded}(x_e) \wedge \text{Jobs}(y_a) \wedge \text{EQ}(u)(z) \\ \wedge \arg_1(x_e, y_a) \wedge \arg_2(x_e, z_a)$$

and *Apple which Jobs founded* has semantics

$$\lambda u.\, \exists xyz.\, \text{founded}(x_e) \wedge \text{Jobs}(y_a) \wedge \text{EQ}(u)(z) \\ \wedge \arg_1(x_e, y_a) \wedge \arg_2(x_e, z_a) \wedge \text{Apple}(u_a)$$

as intended. Note that this latter expression can be simplified, by elimination of the z variable, to

$$\lambda u.\, \exists xy.\, \text{founded}(x_e) \wedge \text{Jobs}(y_a) \\ \wedge \arg_1(x_e, y_a) \wedge \arg_2(x_e, u_a) \wedge \text{Apple}(u_a)$$

Wh Questions. Wh questions are handled using the BIND-mechanism described in the previous section. As one example, the s-expression for *Who did Jim marry* is as follows:

(wh-dobj (BIND f (nsubj (aux (dobj marry f) did)
Jim)) who)

We assume the following lambda expressions:

Who $\Rightarrow \lambda x.\, \text{TARGET}(x_a)$
did $\Rightarrow \lambda x.\, \text{TRUE}$
aux $\Rightarrow \lambda fg.\, f$
wh-dobj $\Rightarrow \lambda fgx.\, f(x) \wedge g(x)$

It can be verified that this gives the final logical form

$$\lambda x.\, \exists yz.\, \text{TARGET}(x_a) \wedge \text{marry}(y_e) \wedge \text{Jim}(z_a) \\ \wedge \arg_1(y_e, z_a) \wedge \arg_2(y_e, x_a)$$

Note that the predicate TARGET is applied to the variable that is the focus of the question. A similar treatment is used for cases with the wh-element in subject position (e.g., *who married Jim*) or where the wh-element is extracted from a prepositional phrase (e.g., *who was Jim married to*).

3.3 Comparison to CCG

In this section we discuss some differences between our approach and CCG-based approaches for mapping sentences to logical forms. Although our focus is on CCG, the arguments are similar for other formalisms that use the lambda calculus in conjunction with a generative grammar, such as HPSG and LFG, or approaches based on context-free grammars.

Our approach differs in two important (and related) respects from CCG: (1) all constituents in our approach have the same semantic type ($\mathbf{Ind} \times \mathbf{Event} \rightarrow \mathbf{Bool}$); (2) our formalism does not make the argument/adjunct distinction, instead essentially treating all modifiers to a given head as adjuncts.

As an example, consider the analysis of *Disney acquired Pixar* within CCG. In this case *acquired* would be assigned the following CCG lexical entry:

$$\text{S}\backslash\text{NP}/\text{NP} \Rightarrow \lambda f_2 f_1 x.\, \exists yz.\, \text{acquired}(x) \wedge f_1(y) \wedge f_2(z) \wedge \text{arg}_1(x, y) \wedge \text{arg}_2(x, z)$$

Note the explicit arguments corresponding to the subject and object of this transitive verb (f_1 and f_2, respectively). An intransitive verb such as *sleeps* would be assigned an entry with a single functional argument corresponding to the subject (f_1):

$$\text{S}\backslash\text{NP} \Rightarrow \lambda f_1 x.\, \exists y.\, \text{sleeps}(x) \wedge f_1(y) \wedge \text{arg}_1(x, y)$$

In contrast, the entries in our system for these two verbs are simply $\lambda x.\, \text{acquired}(x_e)$ and $\lambda x.\, \text{sleeps}(x_e)$. The two forms are similar, have the same semantic type, and do not include variables such as f_1 and f_2 for the subject and object.

The advantage of our approach is that it is robust, and relatively simple, in that a strict grammar that enforces type checking is not required. However, there are challenges in handling some linguistic constructions. A simple example is passive verbs. In our formalism, the passive form of *acquired* has the form $\lambda x.\, \text{acquired.pass}(x_e)$, distinct from its active form $\lambda x.\, \text{acquired}(x_e)$. The sentence *Pixar was acquired* is then assigned the logical form $\lambda x.\, \exists y.\, \text{acquired.pass}(x_e) \wedge \text{Pixar}(y_a) \wedge \text{arg}_1(x_e, y_a)$. Modifying our approach to give the same logical forms for active and passive forms would require a significant extension of our approach. In contrast, in CCG the lexical entry for the passive form of *acquired* can directly specify the mapping between subject position and the arg_2:

$$\text{S}\backslash\text{NP} \Rightarrow \lambda f_2 x.\, \exists z.\, \text{acquired}(x) \wedge f_2(z) \wedge \text{arg}_2(x, z)$$

As another example, correct handling of object and subject control verbs is challenging in the single-type system: for example, in the analysis for *John persuaded Jim to acquire Apple*, the CCG analysis would have an entry for *persuaded* that explicitly takes three arguments (in this case *John*, *Jim*, and *to acquire Apple*) and assigns *Jim* as both the direct object of *persuaded* and as the subject of *acquire*. In our approach the subject relationship to *acquire* is instead recovered in a post-processing step, based on the lexical identity of *persuaded*.

4 Semantic Parsing as Graph Matching

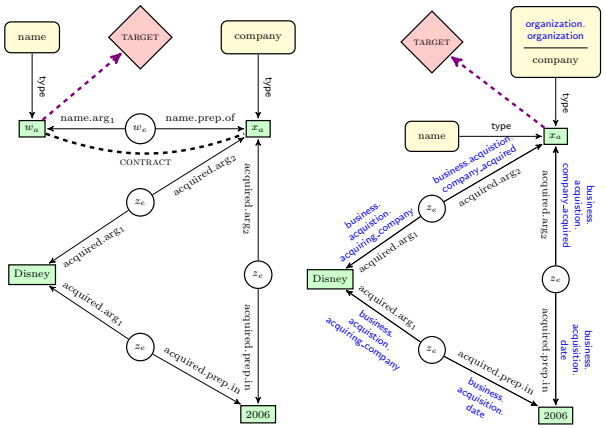
We next describe how the ungrounded logical forms from the previous section are mapped to a fully grounded semantic representation that can be used for question answering against Freebase. Following Reddy et al. (2014), we treat this mapping as a graph matching problem, but instead of deriving ungrounded graphs from CCG-based logical forms, we use the dependency-based logical forms from the previous sections. To learn the mapping to Freebase, we rely on manually assembled question-answer pairs. For each training question, we first find the set of *oracle* grounded graphs—Freebase subgraphs which when executed yield the correct answer—derivable from the question logical form. These oracle graphs are then used to train a structured perceptron model.

4.1 Ungrounded Graphs

We follow Reddy et al. (2014) and first convert logical forms to their corresponding ungrounded graphs. Figure 2(a) shows an example for *What is the name of the company which Disney acquired in 2006?*. Predicates corresponding to resolved entities ($\text{Disney}(y_a)$ and $2006(v_a)$) become entity nodes (rectangles), whereas remaining entity predicates ($\text{name}(w_a)$ and $\text{company}(x_a)$) become entity nodes (w_a and x_a), connected to entity type nodes (name and company; rounded rectangles). The $\text{TARGET}(w_a)$ node (diamond) connects to the entity node whose denotation corresponds to the answer to the question.

4.2 Grounded Graphs

The ungrounded graphs are grounded to Freebase subgraphs by mapping entity nodes, entity-entity



(a) Before CONTRACT. (b) After CONTRACT.

Figure 2: The CONTRACT operation applied to the ungrounded graph for the question *What is the name of the company which Disney acquired in 2006?*. After CONTRACT has been applied the graph is isomorphic to the representation in Freebase; in (b) we show the Freebase predicates after grounding in blue.

edges and entity type nodes in the ungrounded graph to Freebase entities, relations and types, respectively. While Reddy et al. (2014) assume that the ungrounded graphs are isomorphic to their corresponding Freebase subgraph, at least 15% of the examples in our development set do not satisfy this property. For example, the ungrounded graph in Figure 2(a) is not isomorphic to the Freebase subgraph in Figure 2(b), making it impossible to derive the correct grounded graph from the ungrounded one by a direct mapping. To account for such structural mismatch, we introduce two simple transformation operations.

CONTRACT. The CONTRACT operation takes a pair of entity nodes connected by an edge and merges them into a single node. For example, in Figure 2(a) the entity nodes w_a and x_a are connected by an edge via the event w_e. After applying the CONTRACT operation to nodes w_a and x_a, they are merged. Note how in Figure 2(b) all the nodes attached to w_a attach to the node x_a after this operation. The contracted graph is now isomorphic to its Freebase subgraph.

EXPAND. Parse errors may lead to ungrounded graphs with disconnected components. For example, the ungrammatical question *What to do Washington DC December?* results in the lambda expression $\lambda z.\, \exists xyw.\, \text{TARGET}(x_a) \wedge \text{do}(z_e) \wedge \text{arg}_1(z_e, x_a) \wedge \text{Washington\_DC}(y_a) \wedge \text{December}(w_a)$. The corresponding ungrounded graph has three disconnected components (December and Washington_DC, and the component with entity node x_a linked to event z_e). In such cases, the graph is expanded by linking disconnected entity nodes to the event node with the largest edge degree. In the example above, this would add edges corresponding to the predicates $\text{dep}(z_e, y_a) \wedge \text{dep}(z_e, w_a)$, where dep is the predicate introduced by the EXPAND operation when linking y_a and w_a to z_e. When there is no existing event node in the graph, a dummy event node is introduced.

4.3 Learning

We use a linear model to map ungrounded to grounded graphs. The parameters of the model are learned from question-answer pairs. For example, the question *What is the name of the company which Disney acquired in 2006?* is paired with its answer $\{\textit{Pixar}\}$. In line with most work on question answering against Freebase, we do not rely on annotated logical forms associated with the question for training, instead treating grounded graphs as latent variables.

Let q be a question, let u be an ungrounded graph for q and let g be a grounded graph formed by grounding the nodes and edges of u to the knowledge base $\mathcal{K}$ (throughout we use Freebase as the knowledge base). Following Reddy et al. (2014), we use beam search to find the highest scoring pair of ungrounded and grounded graphs $(\hat{u}, \hat{g})$ under the model $\theta \in \Re^n$:

$$(\hat{u}, \hat{g}) = \arg\max_{(u,g)} \theta \cdot \Phi(u, g, q, \mathcal{K}) \,,$$

where $\Phi(u, g, q, \mathcal{K}) \in \Re^n$ denotes the features for the pair of ungrounded and grounded graphs. Note that for a given query there may be multiple ungrounded graphs, primarily due to the optional use of the CONTRACT operation.[3] The feature function has access to the ungrounded and grounded graphs, to the question, as well as to the content of the knowledge base and the denotation $|g|_{\mathcal{K}}$ (the denotation of a grounded graph is defined as the set of entities or attributes reachable at its TARGET node). See Section 5.3 for the features employed.

The model parameters are estimated with the averaged structured perceptron (Collins, 2002; Fre-

[3] Another source of ambiguity may be a lexical item having multiple lambda-calculus entries; in our rules this only arises when analyzing count expressions such as *how many*.

und and Schapire, 1999). Given a training question-answer pair $(q, \mathcal{A})$, the update is:

$$\theta^{t+1} \leftarrow \theta^t + \Phi(u^+, g^+, q, \mathcal{K}) - \Phi(\hat{u}, \hat{g}, q, \mathcal{K}) \,,$$

where (u^+, g^+) denotes the pair of gold ungrounded and grounded graphs for q. Since we do not have direct access to these gold graphs, we instead rely on the set of *oracle graphs*, $\mathcal{O}_{\mathcal{K},\mathcal{A}}(q)$, as a proxy:

$$(u^+, g^+) = \underset{(u,g)\in\mathcal{O}_{\mathcal{K},\mathcal{A}}(q)}{\arg\max} \; \theta^t \cdot \Phi(u, g, q, \mathcal{K}) \,,$$

where $\mathcal{O}_{\mathcal{K},\mathcal{A}}(q)$ is defined as the set of pairs (u, g) derivable from the question q, whose denotation $|g|_{\mathcal{K}}$ has minimal F_1-loss against the gold answer $\mathcal{A}$. We find the oracle graphs for each question a priori by performing beam-search with a beam size of 10k and only use examples with oracle $F_1 > 0.0$ for training.

5 Experimental Setup

We next verify empirically that our proposed approach derives a useful logical compositional semantic representation from dependency syntax. Below, we give details on the evaluation datasets and baselines used for comparison. We also describe the model features and provide implementation details.

5.1 Training and Evaluation Datasets

We evaluated our approach on the Free917 (Cai and Yates, 2013) and WebQuestions (Berant et al., 2013) datasets. Free917 consists of 917 questions manually annotated with their Freebase query. We retrieved the answer to each question by executing its query on Freebase and ignore the query for all subsequent experiments. WebQuestions consists of 5810 question-answer pairs. The standard train/test splits were used for both datasets, with Free917 containing 641 train and 276 test questions and WebQuestions containing 3030 train and 2780 test questions. For all our development experiments we tuned the models on held-out data consisting of 30% of the training questions, while for final testing we used the complete training data.

5.2 Baseline Models and Representations

In addition to the dependency-based semantic representation DEPLAMBDA (Section 3) and previous work on these datasets, we compare to three additional baseline representations outlined below. We use GRAPHPARSER[4] to map these representations to Freebase.

DEPTREE. In this baseline, an ungrounded graph is created directly from the original dependency tree. An event is created for each parent and its dependents in the tree. Each dependent is linked to this event with an edge labeled with its dependency relation, while the parent is linked to the event with an edge labeled $\arg_0$. If a word is a question word, an additional TARGET predicate is attached to its entity node.

SIMPLEGRAPH. This representation has a single event to which all entities in the question are connected by the predicate arg_1. An additional TARGET node is connected to the event by the predicate arg_0. This is similar to the template representation of Yao (2015) and Bast and Haussmann (2015). Note that this cannot represent any compositional structure.

CCGGRAPH. Finally, we compare to the CCG-based semantic representation of Reddy et al. (2014), adding the CONTRACT and EXPAND operations to increase its expressivity.

5.3 Implementation Details

Below are more details of our entity resolution model, the syntactic parser used, features in the grounding model and the beam search procedure.

Entity Resolution. For Free917, we follow prior work and resolve entities by string match against the entity lexicon provided with the dataset. For WebQuestions, we use eight handcrafted part-of-speech patterns to identify entity span candidates. We use the Stanford CoreNLP caseless tagger for part-of-speech tagging (Manning et al., 2014). For each candidate mention span, we retrieve the top 10 entities according to the Freebase API.[5] We then create a lattice in which the nodes correspond to mention-entity pairs, scored by their Freebase API scores, and the edges encode the fact that no joint assignment of entities to mentions can contain overlapping spans. Finally, we generate ungrounded graphs for the top 10 paths through the lattice and treat the final entity disambiguation as part of the semantic parsing problem.

[4] http://github.com/sivareddyg/graph-parser

[5] http://developers.google.com/freebase/

Representation	-C -E	-C +E	+C -E	+C +E
(a) Average oracle F_1				
DEPTREE	30.8	30.8	72.8	72.8
SIMPLEGRAPH	73.0	73.0	73.0	73.0
CCGGRAPH	65.1	70.3	67.6	72.9
DEPLAMBDA	64.8	66.3	71.8	73.0
(b) Average number of oracle graphs per question				
DEPTREE	1.5	1.5	354.6	354.6
SIMPLEGRAPH	1.5	1.5	1.8	1.8
CCGGRAPH	1.6	1.7	3.4	3.4
DEPLAMBDA	1.4	1.5	3.6	4.2
(c) Average F_1				
DEPTREE	19.9	19.9	42.6	42.6
SIMPLEGRAPH	49.0	49.0	48.2	48.2
CCGGRAPH	44.7	47.3	46.5	48.9
DEPLAMBDA	45.9	47.5	48.8	50.4

Table 1: Oracle statistics and accuracies on the WebQuestions development set. +(-)C: with(out) CONTRACT. +(-)E: with(out) EXPAND.

Representation	-C -E	-C +E	+C -E	+C +E
(a) Oracle Accuracy				
DEPTREE	26.0	26.0	95.8	95.8
SIMPLEGRAPH	96.3	96.3	96.3	96.3
CCGGRAPH	91.2	93.3	92.2	95.3
DEPLAMBDA	91.1	92.7	94.3	95.8
(b) Average number of oracle graphs per question				
DEPTREE	1.2	1.2	285.4	285.4
SIMPLEGRAPH	1.6	1.6	1.8	1.8
CCGGRAPH	1.6	1.6	2.4	2.5
DEPLAMBDA	1.5	1.5	3.3	3.4
(c) Accuracy				
DEPTREE	21.3	21.3	51.6	51.6
SIMPLEGRAPH	40.9	40.9	42.0	42.0
CCGGRAPH	68.3	69.4	70.4	71.0
DEPLAMBDA	69.3	71.3	72.4	73.4

Table 2: Oracle statistics and accuracies on the Free917 development set. +(-)C: with(out) CONTRACT. +(-)E: with(out) EXPAND.

Syntactic Parsing. We recase the resolved entity mentions and run a case-sensitive second-order conditional random field part-of-speech tagger (Lafferty et al., 2001). The hypergraph parser of Zhang and McDonald (2014) is used for dependency parsing. The tagger and parser are both trained on the OntoNotes 5.0 corpus (Weischedel et al., 2011), with constituency trees converted to Stanford-style dependencies (De Marneffe and Manning, 2008). To derive the CCG-based representation, we use the output of the EasyCCG parser (Lewis and Steedman, 2014).

Features. We use the features from Reddy et al. (2014), which include edge alignment and stem overlap between ungrounded and grounded graphs, and contextual features such as word and grounded relation pairs. In addition, we introduce a feature indicating the use of the CONTRACT operation: (*MergedSubEdge, HeadSubEdge, MergedIsEntity, HeadIsEntity*). For example, in Figure 2 the edge between w_a and x_a is contracted to x_a, resulting in the feature (name.arg_1, name.prep.of, False, False). The EXPAND operation is treated as a pre-processing step and no features are used to encode its use. Finally, the entity-lattice score is used as a real valued feature.

Beam Search. We use beam search to infer the highest scoring graph pair for a question. The search operates over entity-entity edges and entity type nodes of each ungrounded graph. For an entity-entity edge, we can ground the edge to a Freebase relation, contract the edge in either direction, or skip the edge. For an entity type node, we can ground the node to a Freebase type, or skip the node. The order of traversal is based on the number of named entities connected to an edge. After an edge is grounded, the entity type nodes connected to it are grounded in turn, before the next edge is processed. To restrict the search, if two beam items correspond to the same grounded graph, the one with the lower score is discarded. A beam size of 100 was used in all experiments.

6 Experimental Results

We examine the different representations for question answering along two axes. First, we compare their expressiveness in terms of answer reachability assuming a perfect model. Second, we compare their performance with a learned model. Finally, we conduct a detailed error analysis of DEPLAMBDA, with a comparison to the errors made by CCGGRAPH. For WebQuestions evaluation is in terms of the average F_1-score across questions, while for Free917, evaluation is in terms of exact answer accuracy.[6]

6.1 Expressiveness of the Representations

Table 1(a) and Table 2(a) show the oracle F_1-scores of each representation on the WebQuestions and

[6]We use the evaluation scripts available at http://www-nlp.stanford.edu/software/sempre and http://github.com/elmar-haussmann/aqqu, respectively.

Free917 development sets respectively. According to the first column (-C -E), the original DEPTREE representation can be directly mapped to Freebase for less than a third of the questions. Adding the CONTRACT operation (+C) improves this substantially to an oracle F_1 of about 73% on WebQuestions and 95.8% on Free917. However, this comes at the cost of massive spurious ambiguity: from Table 1(b) there are on average over 300 oracle graphs for a single dependency tree. Table 1(c) shows the results of the different representations on the WebQuestions development set. Spurious ambiguity clearly hampers learning and DEPTREE falls behind the other representations. CCGGRAPH and DEPLAMBDA align much more closely to Freebase and achieve similar oracle F_1 scores with far less spurious ambiguity. SIMPLEGRAPH, which cannot represent any compositional semantics, is competitive with these syntax-based representations. This might come as a surprise, but it simply reflects the fact that the dataset does not contain questions that require compositional reasoning.

6.2 Results on WebQuestions and Free917

We use the best settings on the development set in subsequent experiments, i.e., with CONTRACT and EXPAND enabled. Table 3 shows the results on the WebQuestions and Free917 test sets with additional entries for recent prior work on these datasets. The trend from the development set carries over and DEPLAMBDA outperforms the other graph-based representations, while performing slightly below the state-of-the-art model of Yih et al. (2015) ("Y&C"), which uses a separately trained entity resolution system (Yang and Chang, 2015). When using the standard Freebase API ("FB API") for entity resolution, the performance of their model drops to 48.4% F_1.

On Free917, DEPLAMBDA outperforms all other representations by a wide margin and obtains the best result to date. Interestingly, DEPTREE outperforms SIMPLEGRAPH in this case. We attribute this to the small training set and larger lexical variation of Free917. The structural features of the graph-based representations seem highly beneficial in this case.

6.3 Error Analysis

We categorized 100 errors made by DEPLAMBDA (+C +E) on the WebQuestions development set. In 43 cases the correct answer is present in the beam,

Method	Free917 Accuracy	WebQuestions Average F_1
Cai and Yates (2013)	59.0	–
Berant et al. (2013)	62.0	35.7
Kwiatkowski et al. (2013)	68.0	–
Yao and Van Durme (2014)	–	33.0
Berant and Liang (2014)	68.5	39.9
Bao et al. (2014)	–	37.5
Bordes et al. (2014)	–	39.2
Yao (2015)	–	44.3
Yih et al. (2015) (FB API)	–	48.4
Bast and Haussmann (2015)	76.4	49.4
Berant and Liang (2015)	–	49.7
Yih et al. (2015) (Y&C)	–	**52.5**
This Work		
DEPTREE	53.2	40.4
SIMPLEGRAPH	43.7	48.5
CCGGRAPH (+C +E)	73.3	48.6
DEPLAMBDA (+C +E)	**78.0**	50.3

Table 3: Question-answering results on the WebQuestions and Free917 test sets.

but ranked below an incorrect answer (e.g., for *where does volga river start*, the annotated gold answer is *Valdai Hills*, which is ranked second, with *Russia*, *Europe* ranked first). In 35 cases, only a subset of the answer is predicted correctly (e.g, for *what countries in the world speak german*, the system predicts *Germany* from the human_language.main_country Freebase relation, whereas the gold relation human_language.countries_spoken_in gives multiple countries). Together, these two categories correspond to roughly 80% of the errors. In 10 cases, the Freebase API fails to add the gold entity to the lattice (e.g., for *who is blackwell*, the correct *blackwell* entity was missing). Due to the way WebQuestions was crowdsourced, 9 questions have incorrect or incomplete gold annotations (e.g., *what does each fold of us flag means* is answered with *USA*). The remaining 3 cases are due to structural mismatch (e.g., in *who is the new governor of florida 2011*, the graph failed to connect the target node with both *2011* and *Florida*).

Due to the ungrammatical nature of WebQuestions, CCGGRAPH fails to produce ungrounded graphs for 4.5% of the complete development set, while DEPLAMBDA is more robust with only 0.9% such errors. The CCG parser is restricted to produce a sentence tag as the final category in the syntactic derivation, which penalizes ungrammatical analyses (e.g., *what*

victoria beckham kids names and *what nestle owns*). Examples where DEPLAMBDA fails due to parse errors, but CCGGRAPH succeed include *when was blessed kateri born* and *where did anne frank live before the war*. Note that the EXPAND operation mitigates some of these problems. While CCG is known for handling comparatives elegantly (e.g., *who was sworn into office when john f kennedy was assassinated*), we do not have a special treatment for them in the semantic representation. Differences in syntactic parsing performance and the somewhat limited expressivity of the semantic representation are likely the reasons for CCGGRAPH's lower performance.

7 Related Work

There are two relevant strands of prior work: general purpose ungrounded semantics and grounded semantic parsing. The former have been studied on their own and as a component in tasks such as semantic parsing to knowledge bases (Kwiatkowski et al., 2013; Reddy et al., 2014; Choi et al., 2015; Krishnamurthy and Mitchell, 2015), sentence simplification (Narayan and Gardent, 2014), summarization (Liu et al., 2015), paraphrasing (Pavlick et al., 2015) and relation extraction (Rocktäschel et al., 2015). There are two ways of generating these representations: either relying on syntactic structure and producing the semantics post hoc, or generating it directly from text. We adopt the former approach, which was pioneered by Montague (1973) and is becoming increasingly attractive with the advent of accurate syntactic parsers.

There have been extensive studies on extracting semantics from syntactic representations such as LFG (Dalrymple et al., 1995), HPSG (Copestake et al., 2001; Copestake et al., 2005), TAG (Gardent and Kallmeyer, 2003; Joshi et al., 2007) and CCG (Baldridge and Kruijff, 2002; Bos et al., 2004; Steedman, 2012; Artzi et al., 2015). However, few have used dependency structures for this purpose. Debusmann et al. (2004) and Cimiano (2009) describe grammar-based conversions of dependencies to semantic representations, but do not validate them empirically. Stanovsky et al. (2016) use heuristics based on linguistic grounds to convert dependencies to proposition structures. Bédaride and Gardent (2011) propose a graph-rewriting technique to convert a graph built from dependency trees and semantic role structures to a first-order logical form, and present results on textual entailment. Our work, in contrast, assumes access only to dependency trees and offers an alternative method based on the lambda calculus, mimicking the structure of knowledge bases such as Freebase; we further present extensive empirical results on recent question-answering corpora.

Structural mismatch between the source semantic representation and the target application's representation is an inherent problem with approaches using general-purpose representations. Kwiatkowski et al. (2013) propose lambda-calculus operations to generate multiple type-equivalent expressions to handle this mismatch. In contrast, we use graph-transduction operations which are relatively easier to interpret. There is also growing work on converting syntactic structures to the target application's structure without going through an intermediate semantic representation, e.g., answer-sentence selection (Punyakanok et al., 2004; Heilman and Smith, 2010; Yao et al., 2013) and semantic parsing (Ge and Mooney, 2009; Poon, 2013; Parikh et al., 2015; Xu et al., 2015; Wang et al., 2015; Andreas and Klein, 2015).

A different paradigm is to directly parse the text into a grounded semantic representation. Typically, an over-generating grammar is used whose accepted parses are ranked (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Wong and Mooney, 2007; Kwiatkowksi et al., 2010; Liang et al., 2011; Berant et al., 2013; Flanigan et al., 2014; Groschwitz et al., 2015). In contrast, Bordes et al. (2014) and Dong et al. (2015) discard the notion of a target representation altogether and instead learn to rank potential answers to a given question by embedding questions and answers into the same vector space.

8 Conclusion

We have introduced a method for converting dependency structures to logical forms using the lambda calculus. A key idea of this work is the use of a single semantic type for every constituent of the dependency tree, which provides us with a robust way of compositionally deriving logical forms. The resulting representation is subsequently grounded to Freebase by learning from question-answer pairs. Empirically, the proposed representation was shown to be superior to the original dependency trees and more robust than logical forms derived from a CCG parser.

Acknowledgements

This work greatly benefitted from discussions with Slav Petrov, John Blitzer, Fernando Pereira, Emily Pitler and Nathan Schneider. The authors would also like to thank Christopher Potts and the three anonymous reviewers for their valuable feedback. We acknowledge the financial support of EU IST Cognitive Systems IP EC-FP7-270273 "Xperience" (Steedman) and EPSRC (EP/K017845/1) in the framework of the CHIST-ERA READERS project (Lapata).

References

Jacob Andreas and Dan Klein. 2015. Alignment-Based Compositional Semantics for Instruction Following. In *Proceedings of Empirical Methods on Natural Language Processing*, pages 1165–1174.

Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage CCG Semantic Parsing with AMR. In *Proceedings of Empirical Methods on Natural Language Processing*, pages 1699–1710.

Jason Baldridge and Geert-Jan Kruijff. 2002. Coupling CCG and Hybrid Logic Dependency Semantics. In *Proceedings of Association for Computational Linguistics*, pages 319–326.

Junwei Bao, Nan Duan, Ming Zhou, and Tiejun Zhao. 2014. Knowledge-Based Question Answering as Machine Translation. In *Proceedings of Association for Computational Linguistics*, pages 967–976.

Hannah Bast and Elmar Haussmann. 2015. More Accurate Question Answering on Freebase. In *Proceedings of ACM International Conference on Information and Knowledge Management*, pages 1431–1440.

Paul Bédaride and Claire Gardent. 2011. Deep Semantics for Dependency Structures. In *Proceedings of Conference on Intelligent Text Processing and Computational Linguistics*, pages 277–288.

Jonathan Berant and Percy Liang. 2014. Semantic Parsing via Paraphrasing. In *Proceedings of Association for Computational Linguistics*, pages 1415–1425.

Jonathan Berant and Percy Liang. 2015. Imitation Learning of Agenda-Based Semantic Parsers. *Transactions of the Association for Computational Linguistics*, 3:545–558.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proceedings of Empirical Methods on Natural Language Processing*, pages 1533–1544.

Antoine Bordes, Sumit Chopra, and Jason Weston. 2014. Question Answering with Subgraph Embeddings. In *Proceedings of Empirical Methods on Natural Language Processing*, pages 615–620.

Johan Bos, Stephen Clark, Mark Steedman, James R. Curran, and Julia Hockenmaier. 2004. Wide-Coverage Semantic Representations from a CCG Parser. In *Proceedings of International Conference on Computational Linguistics*, pages 1240–1246.

Qingqing Cai and Alexander Yates. 2013. Large-scale Semantic Parsing via Schema Matching and Lexicon Extension. In *Proceedings of Association for Computational Linguistics*, pages 423–433.

Bob Carpenter. 1998. *Type-Logical Semantics*. MIT Press, Cambridge, MA, USA.

David L. Chen and Raymond J. Mooney. 2011. Learning to Interpret Natural Language Navigation Instructions from Observations. In *Proceedings of Association for the Advancement of Artificial Intelligence*, pages 1–2.

Eunsol Choi, Tom Kwiatkowski, and Luke Zettlemoyer. 2015. Scalable Semantic Parsing with Partial Ontologies. In *Proceedings of Association for Computational Linguistics*, pages 1311–1320.

Philipp Cimiano. 2009. Flexible Semantic Composition with DUDES. In *Proceedings of International Conference on Computational Semantics*, pages 272–276.

Michael Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of Empirical Methods on Natural Language Processing*, pages 1–8.

Ann Copestake, Alex Lascarides, and Dan Flickinger. 2001. An Algebra for Semantic Construction in Constraint-based Grammars. In *Proceedings of Association for Computational Linguistics*, pages 140–147.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A. Sag. 2005. Minimal Recursion Semantics: An Introduction. *Research on Language and Computation*, 3(2-3):281–332.

Mary Dalrymple, John Lamping, Fernando C. N. Pereira, and Vijay A. Saraswat. 1995. Linear Logic for Meaning Assembly. In *Proceedings of Computational Logic for Natural Language Processing*.

Marie-Catherine De Marneffe and Christopher D Manning. 2008. Stanford typed dependencies manual. Technical report, Stanford University.

Ralph Debusmann, Denys Duchier, Alexander Koller, Marco Kuhlmann, Gert Smolka, and Stefan Thater. 2004. A Relational Syntax-Semantics Interface Based on Dependency Grammar. In *Proceedings of International Conference on Computational Linguistics*, pages 176–182.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2015. Question Answering over Freebase with Multi-Column Convolutional Neural Networks. In *Proceedings of Association for Computational Linguistics*, pages 260–269.

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A. Smith. 2014. A Discriminative Graph-Based Parser for the Abstract Meaning Representation. In *Proceedings of Association for Computational Linguistics*, pages 1426–1436.

Yoav Freund and Robert E. Schapire. 1999. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37(3):277–296, December.

Claire Gardent and Laura Kallmeyer. 2003. Semantic Construction in Feature-based TAG. In *Proceedings of European Chapter of the Association for Computational Linguistics*, pages 123–130.

Ruifang Ge and Raymond Mooney. 2009. Learning a Compositional Semantic Parser using an Existing Syntactic Parser. In *Proceedings of Association for Computational Linguistics*, pages 611–619.

Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. 2015. Graph parsing with s-graph grammars. In *Proceedings of Association for Computational Linguistics*, pages 1481–1490.

Michael Heilman and Noah A. Smith. 2010. Tree Edit Models for Recognizing Textual Entailments, Paraphrases, and Answers to Questions. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pages 1011–1019.

Aravind K. Joshi, Laura Kallmeyer, and Maribel Romero. 2007. Flexible Composition In LTAG: Quantifier Scope and Inverse Linking. In Harry Bunt and Reinhard Muskens, editors, *Computing Meaning*, volume 83 of *Studies in Linguistics and Philosophy*, pages 233–256. Springer Netherlands.

Jayant Krishnamurthy and Tom Mitchell. 2012. Weakly Supervised Training of Semantic Parsers. In *Proceedings of Empirical Methods on Natural Language Processing*, pages 754–765.

Jayant Krishnamurthy and Tom M. Mitchell. 2015. Learning a Compositional Semantics for Freebase with an Open Predicate Vocabulary. *Transactions of the Association for Computational Linguistics*, 3:257–270.

Tom Kwiatkowksi, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing Probabilistic CCG Grammars from Logical Form with Higher-Order Unification. In *Proceedings of Empirical Methods on Natural Language Processing*, pages 1223–1233.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. 2013. Scaling Semantic Parsers with On-the-Fly Ontology Matching. In *Proceedings of Empirical Methods on Natural Language Processing*, pages 1545–1556.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of International Conference on Machine Learning*, pages 282–289.

Mike Lewis and Mark Steedman. 2014. A* CCG Parsing with a Supertag-factored Model. In *Proceedings of Empirical Methods on Natural Language Processing*, pages 990–1000.

Percy Liang, Michael Jordan, and Dan Klein. 2011. Learning Dependency-Based Compositional Semantics. In *Proceedings of Association for Computational Linguistics*, pages 590–599.

Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman Sadeh, and Noah A. Smith. 2015. Toward Abstractive Summarization Using Semantic Representations. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pages 1077–1086.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of Association for Computational Linguistics*, pages 55–60.

Andre Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the Turbo: Fast Third-Order Non-Projective Turbo Parsers. In *Proceedings of Association for Computational Linguistics*, pages 617–622.

Cynthia Matuszek, Nicholas FitzGerald, Luke Zettlemoyer, Liefeng Bo, and Dieter Fox. 2012. A Joint Model of Language and Perception for Grounded Attribute Learning. In *Proceedings of International Conference on Machine Learning*.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-Projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of Empirical Methods on Natural Language Processing*, pages 523–530.

Richard Montague. 1973. The Proper Treatment of Quantification in Ordinary English. In K.J.J. Hintikka, J.M.E. Moravcsik, and P. Suppes, editors, *Approaches to Natural Language*, volume 49 of *Synthese Library*, pages 221–242. Springer Netherlands.

Michael Moortgat. 1988. *Categorical Investigations. Logical and Linguistic Aspects of the Lambek Calculus*. Foris, Dordrecht.

Michael Moortgat. 1991. Generalized Quantification and Discontinuous Type Constructors. Technical report, University of Utrecht.

Shashi Narayan and Claire Gardent. 2014. Hybrid Simplification using Deep Semantics and Machine Translation. In *Proceedings of Association for Computational Linguistics*, pages 435–445.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. MaltParser: A Language-Independent System for Data-Driven Dependency Parsing. *Natural Language Engineering*, 13(2):95–135.

Ankur P. Parikh, Hoifung Poon, and Kristina Toutanova. 2015. Grounded Semantic Parsing for Complex Knowledge Extraction. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pages 756–766.

Ellie Pavlick, Johan Bos, Malvina Nissim, Charley Beller, Benjamin Van Durme, and Chris Callison-Burch. 2015. Adding Semantics to Data-Driven Paraphrasing. In *Proceedings of Association for Computational Linguistics*, pages 1512–1522.

Fernando C. N. Pereira. 1990. Categorial Semantics and Scoping. *Computational Linguistics*, 16(1):1–10.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Hoifung Poon. 2013. Grounded Unsupervised Semantic Parsing. In *Proceedings of Association for Computational Linguistics*, pages 933–943.

Vasin Punyakanok, Dan Roth, and Wen-tau Yih. 2004. Mapping Dependencies Trees: An Application to Question Answering. In *Proceedings of International Symposium on Artificial Intelligence and Mathematics*, pages 1–10.

Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale Semantic Parsing without Question-Answer Pairs. *Transactions of the Association for Computational Linguistics*, 2:377–392.

Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. 2015. Injecting Logical Background Knowledge into Embeddings for Relation Extraction. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pages 1119–1129.

Gabriel Stanovsky, Jessica Ficler, Ido Dagan, and Yoav Goldberg. 2016. Getting More Out Of Syntax with PropS. *ArXiv e-prints*, March.

Mark Steedman. 2012. *Taking Scope - The Natural Semantics of Quantifiers*. MIT Press.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015. A Transition-based Algorithm for AMR Parsing. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pages 366–375.

Ralph Weischedel, Eduard Hovy, Martha Palmer, Mitch Marcus, Robert Belvin, Sameer Pradhan, Lance Ramshaw, and Nianwen Xue. 2011. OntoNotes: A Large Training Corpus for Enhanced Processing. In J. Olive, C. Christianson, and J. McCary, editors, *Handbook of Natural Language Processing and Machine Translation*. Springer.

Yuk Wah Wong and Raymond J. Mooney. 2006. Learning for Semantic Parsing with Statistical Machine Translation. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pages 439–446.

Yuk Wah Wong and Raymond Mooney. 2007. Learning Synchronous Grammars for Semantic Parsing with Lambda Calculus. In *Proceedings of Association for Computational Linguistics*, pages 960–967.

Kun Xu, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2015. Question Answering via Phrasal Semantic Parsing. In *Proceedings of Conference and Labs of the Evaluation Forum*, pages 414–426.

Yi Yang and Ming-Wei Chang. 2015. S-MART: Novel Tree-based Structured Learning Algorithms Applied to Tweet Entity Linking. In *Proceedings of Association for Computational Linguistics*, pages 504–513.

Xuchen Yao and Benjamin Van Durme. 2014. Information Extraction over Structured Data: Question Answering with Freebase. In *Proceedings of Association for Computational Linguistics*, pages 956–966.

Xuchen Yao, Benjamin Van Durme, Chris Callison-Burch, and Peter Clark. 2013. Answer Extraction as Sequence Tagging with Tree Edit Distance. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pages 858–867.

Xuchen Yao. 2015. Lean Question Answering over Freebase from Scratch. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pages 66–70.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base. In *Proceedings of Association for Computational Linguistics*, pages 1321–1331.

John M. Zelle and Raymond J. Mooney. 1996. Learning to Parse Database Queries Using Inductive Logic Programming. In *Proceedings of Association for the Advancement of Artificial Intelligence*, pages 1050–1055.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 658–666.

Hao Zhang and Ryan McDonald. 2014. Enforcing Structural Diversity in Cube-pruned Dependency Parsing. In *Proceedings of Association for Computational Linguistics*, pages 656–661.