Grounded Unsupervised Semantic Parsing

Hoifung Poon
One Microsoft Way
Microsoft Research
Redmond, WA 98052, USA
hoifung@microsoft.com

Abstract

We present the first unsupervised approach for semantic parsing that rivals the accuracy of supervised approaches in translating natural-language questions to database queries. Our GUSP system produces a semantic parse by annotating the dependency-tree nodes and edges with latent states, and learns a probabilistic grammar using EM. To compensate for the lack of example annotations or question-answer pairs, GUSP adopts a novel grounded-learning approach to leverage database for indirect supervision. On the challenging ATIS dataset, GUSP attained an accuracy of 84%, effectively tying with the best published results by supervised approaches.

1 Introduction

Semantic parsing maps text to a formal meaning representation such as logical forms or structured queries. Recently, there has been a burgeoning interest in developing machine-learning approaches for semantic parsing (Zettlemoyer and Collins, 2005; Zettlemoyer and Collins, 2007; Mooney, 2007; Kwiatkowski et al., 2011), but the predominant paradigm uses supervised learning, which requires example annotations that are costly to obtain. More recently, several grounded-learning approaches have been proposed to alleviate the annotation burden (Chen and Mooney, 2008; Kim and Mooney, 2010; Börschinger et al., 2011; Clarke et al., 2010; Liang et al., 2011). In particular, Clarke et al. (2010) and Liang et al. (2011) proposed methods to learn from question-answer pairs alone, which represents a significant advance. However, although these methods exonerate annotators from mastering specialized logical forms, finding the answers for complex questions still requires non-trivial effort. [1]

Poon & Domingos (2009, 2010) proposed the USP system for unsupervised semantic parsing, which learns a parser by recursively clustering and composing synonymous expressions. While their approach completely obviates the need for direct supervision, their target logic forms are self-induced clusters, which do not align with existing database or ontology. As a result, USP can not be used directly to answer complex questions against an existing database. More importantly, it misses the opportunity to leverage database for indirect supervision.

In this paper, we present the GUSP system, which combines unsupervised semantic parsing with grounded learning from a database. GUSP starts with the dependency tree of a sentence and produces a semantic parse by annotating the nodes and edges with latent semantic states derived from the database. Given a set of natural-language questions and a database, GUSP learns a probabilistic semantic grammar using EM. To compensate for the lack of direct supervision, GUSP constrains the search space using the database schema, and bootstraps learning using lexical scores computed from the names and values of database elements.

Unlike previous grounded-learning approaches, GUSP does not require ambiguous annotations or oracle answers, but rather focuses on leveraging database contents that are readily available. Unlike USP, GUSP predetermines the target logical forms based on the database schema, which alleviates the difficulty in learning and ensures that the output semantic parses can be directly used in querying the database. To handle syntax-semantics mismatch, GUSP introduces a novel dependency-based meaning representation

[1] Clarke et al. (2010) and Liang et al. (2011) used the annotated logical forms to compute answers for their experiments.

by augmenting the state space to represent semantic relations beyond immediate dependency neighborhood. This representation also factorizes over nodes and edges, enabling linear-time exact inference in GUSP.

We evaluated GUSP on end-to-end question answering using the ATIS dataset for semantic parsing (Zettlemoyer and Collins, 2007). Compared to other standard datasets such as GEO and JOBS, ATIS features a database that is an order of magnitude larger in the numbers of relations and instances, as well as a more irregular language (ATIS questions were derived from spoken dialogs). Despite these challenges, GUSP attains an accuracy of 84% in end-to-end question answering, effectively tying with the state-of-the-art supervised approaches (85% by Zettlemoyer & Collins (2007), 83% by Kwiatkowski et al. (2011)).

2 Background

2.1 Semantic Parsing

The goal of semantic parsing is to map text to a complete and detailed meaning representation (Mooney, 2007). This is in contrast with semantic role labeling (Carreras and Marquez, 2004) and information extraction (Banko et al., 2007; Poon and Domingos, 2007), which have a more restricted goal of identifying local semantic roles or extracting selected information slots.

The standard language for meaning representation is first-order logic or a sublanguage, such as FunQL (Kate et al., 2005; Clarke et al., 2010) and lambda calculus (Zettlemoyer and Collins, 2005; Zettlemoyer and Collins, 2007). Poon & Domingos (2009, 2010) induce a meaning representation by clustering synonymous lambda-calculus forms stemming from partitions of dependency trees. More recently, Liang et al. (2011) proposed DCS for dependency-based compositional semantics, which represents a semantic parse as a tree with nodes representing database elements and operations, and edges representing relational joins.

In this paper, we focus on semantic parsing for natural-language interface to database (Grosz et al., 1987). In this problem setting, a natural-language question is first translated into a meaning representation by semantic parsing, and then converted into a structured query such as SQL to obtain answer from the database.

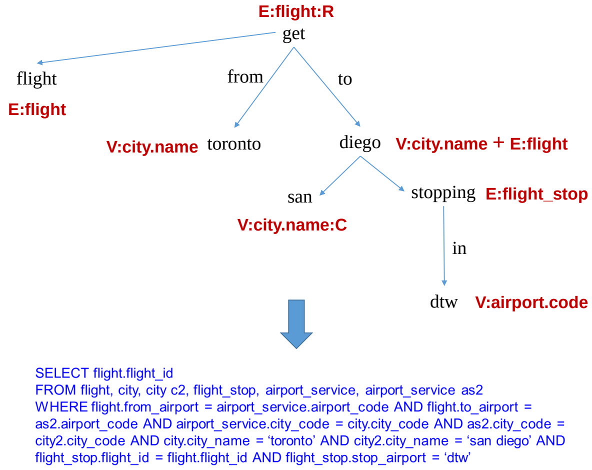
2.2 Unsupervised Semantic Parsing

Unsupervised semantic parsing was first proposed by Poon & Domingos (2009, 2010) with their USP system. USP defines a probabilistic model over the dependency tree and semantic parse using Markov logic (Domingos and Lowd, 2009), and recursively clusters and composes synonymous dependency treelets using a hard EM-like procedure. Since USP uses nonlocal features (e.g., the argument-number feature) and operates over partitions, exact inference is intractable, and USP resorts to a greedy approach to find the MAP parse by searching over partitions. Titov & Klementiev (2011) proposed a Bayesian version of USP and Titov & Klementiev (2012) adapted it for semantic role induction. In USP, the meaning is represented by self-induced clusters. Therefore, to answer complex questions against a database, it requires an additional ontology matching step to resolve USP clusters with database elements.

Popescu et al. (2003, 2004) proposed the PRECISE system, which does not require labeled examples and can be directly applied to question answering with a database. The PRECISE system, however, requires substantial amount of engineering, including a domain-specific lexicon that specifies the synonyms for names and values of database elements, a restricted set of potential interpretations for domain verbs and prepositions, as well as a set of domain questions with manually labeled POS tags for retraining the tagger and parser. It also focuses on the subset of easy questions ("semantically tractable" questions), and sidesteps the problem of dealing with complex and nested structures, as well as ambiguous interpretations. Remarkably, while PRECISE can be very accurate on easy questions, it does not try to learn from these interpretations. In contrast, Goldwasser et al. (2011) proposed a self-supervised approach, which iteratively chose high-confidence parses to retrain the parser. Their system, however, still required a lexicon manually constructed for the given domain. Moreover, it was only applied to a small domain (a subset of GEO), and the result still trailed supervised systems by a wide margin.

2.3 Grounded Learning for Semantic Parsing

Grounded learning is motivated by alleviating the burden of direct supervision via interaction with the world, where the indirect supervision may take the form as ambiguous annotations (Chen



```
SELECT flight.flight_id
FROM flight, city, city c2, flight_stop, airport_service, airport_service as2
WHERE flight.from_airport = airport_service.airport_code AND flight.to_airport =
as2.airport_code AND airport_service.city_code = city.city_code AND as2.city_code =
city2.city_code AND city.city_name = ‘toronto’ AND city2.city_name = ‘san diego’ AND
flight_stop.flight_id = flight.flight_id AND flight_stop.stop_airport = ‘dtw’
```

Figure 1: End-to-end question answering by GUSP for sentence *get flight from toronto to san diego stopping in dtw*. Top: the dependency tree of the sentence is annotated with latent semantic states by GUSP. For brevity, we omit the edge states. Raising occurs from *flight* to *get* and sinking occurs from *get* to *diego*. Bottom: the semantic tree is deterministically converted into SQL to obtain answer from the database.

and Mooney, 2008; Kim and Mooney, 2010; Börschinger et al., 2011) or example question-answer pairs (Clarke et al., 2010; Liang et al., 2011). In general, however, such supervision is not always available or easy to obtain. In contrast, databases are often abundantly available, especially for important domains.

The database community has considerable amount of work on leveraging databases in various tasks such as entity resolution, schema matching, and others. To the best of our knowledge, this approach is still underexplored in the NLP community. One notable exception is distant supervision (Mintz et al., 2009; Riedel et al., 2010; Hoffmann et al., 2011; Krishnamurthy and Mitchell, 2012; Heck et al., 2013), which used database instances to derive training examples for relation extraction. This approach, however, still has considerable limitations. For example, it only handles binary relations, and the quality of the training examples is inherently noisy and hard to control. Moreover, this approach is not applicable to the question-answering setting considered in this paper, since entity pairs in questions need not correspond to valid relational instances in the database.

3 Grounded Unsupervised Semantic Parsing

In this section, we present the GUSP system for grounded unsupervised semantic parsing. GUSP is unsupervised and does not require example logical forms or question-answer pairs. Figure 1 shows an example of end-to-end question answering using GUSP. GUSP produces a semantic parse of the question by annotating its dependency tree with latent semantic states. The semantic tree can then be deterministically converted into SQL to obtain answer from the database. Given a set of natural-language questions and a database, GUSP learns a probabilistic semantic grammar using EM.

To compensate for the lack of annotated examples, GUSP derives indirect supervision from a novel combination of three key sources. First, GUSP leverages the target database to constrain the search space. Specifically, it defines the semantic states based on the database schema, and derives lexical-trigger scores from database elements to bootstrap learning.

Second, in contrast to most existing approaches for semantic parsing, GUSP starts directly from dependency trees and focuses on translating them into semantic parses. While syntax may not always align perfectly with semantics, it is still highly informative about the latter. In particular, dependency edges are often indicative of semantic relations. On the other hand, syntax and semantic often diverge, and synactic parsing errors abound. To combat this problem, GUSP introduces a novel dependency-based meaning representation with an augmented state space to account for semantic relations that are nonlocal in the dependency tree.

GUSP's approach of starting directly from dependency tree is inspired by USP. However, GUSP uses a different meaning representation defined over individual nodes and edges, rather than partitions, which enables linear-time exact inference. GUSP also handles complex linguistic phenomena and syntax-semantics mismatch by explicitly augmenting the state space, whereas USP's capability in handling such phenomena is indirect and more limited.

GUSP represents meaning by a semantic tree, which is similar to DCS (Liang et al., 2011). Their approach to semantic parsing, however, differs from GUSP in that it induced the semantic tree directly from a sentence, rather than starting from a dependency tree and annotating it. Their approach alleviates some complexity in the meaning representation for handling syntax-semantics mismatch, but it has to search over a much larger search space involving exponentially many candidate trees. This might partially explain why it has not yet been scaled up to the ATIS dataset.

Finally, GUSP recognizes that certain aspects in semantic parsing may not be worth learning using precious annotated examples. These are domain-independent and closed-class expressions, such as times and dates (e.g., *before 5pm* and *July seventeenth*), logical connectives (e.g., *and*, *or*, *not*), and numerics (e.g., *200 dollars*). GUSP preprocesses the text to detect such expressions and restricts their interpretation to database elements of compatible types (e.g., *before 5pm* vs. `flight.departure_time` or `flight.arrival_time`). Short of training examples, GUSP also resolves quantifier scoping ambiguities deterministically by a fixed ordering. For example, in the phrase *cheapest flight to Seattle*, the scope of *cheapest* can be either *flight* or *flight to seattle*. GUSP always chooses to apply the superlative at last, amounting to choosing the most restricted scope (*flight to seattle*), which is usually the correct interpretation.

In the remainder of this section, we first formalize the problem setting and introduce the GUSP meaning representation. We then present the GUSP model and learning and inference algorithms. Finally, we describe how to convert a GUSP semantic parse into SQL.

3.1 Problem Formulation

Let d be a dependency tree, $N(d)$ and $E(d)$ be its nodes and edges. In GUSP, a semantic parse of d is an assignment $z : N(d) \cup E(d) \rightarrow \mathcal{S}$ that maps its nodes and edges to semantic states in $\mathcal{S}$. For example, in the example in Figure 1, $z(flight) = \texttt{E:flight}$. At the core of GUSP is a joint probability distribution $P_\theta(d, z)$ over the dependency tree and the semantic parse. Semantic parsing in GUSP amounts to finding the most probable parse $z^* = \arg\max_z P_\theta(d, z)$. Given a set of sentences and their dependency trees D, learning in GUSP maximizes the log-likelihood of D while summing out the latent parses z:

$$\begin{aligned}\theta^* &= \arg\max \log P_\theta(D)\\ &= \arg\max \sum_{d \in D} \log \sum_z P_\theta(d, z)\end{aligned}$$

3.2 Simple Semantic States

Node states GUSP creates a state `E:X` (`E` short for entity) for each database entity X (i.e., a database table), a state `P:Y` (`P` short for property) and `V:Y` (`V` short for value) for each database attribute Y (i.e., a database column). Node states are assigned to dependency nodes. Intuitively, they represent database entities, properties, and values. For example, the ATIS domain contains entities such as `flight` and `fare`, which may contain properties such as the departure time `flight.departure_time` or ticket price `fare.one_direction_cost`. The mentions of entities and properties are represented by entity and property states, whereas constants such as `9:25am` or `120 dollars` are represented by value states. In the semantic parse in Figure 1, for example, *flight* is assigned to entity state `E:flight`, where *toronto* is assigned to value state `V:city.name`. There is a special node state `NULL`, which signifies that the subtree headed by the word contributes no meaning to the semantic parse (e.g., an auxilliary verb).

Edge states GUSP creates an edge state for each valid relational join paths connecting two node states. Edge states are assigned to dependency edges. GUSP enforces the constraints that the node states of the dependency parent and child must agree with the node states in the edge state. For example, `E:flight-V:flight.departure_time` represents a natural join between the flight entity and the property value departure time. For a dependency edge $e : a \rightarrow b$, the assignment to `E:flight-V:flight.departure_time` signifies that a represents a flight entity, and b represents the value of its departure time. An edge state may also represent a relational path consisting of a serial of joins. For example, Zettlemoyer and Collins (2007) used a predicate `from(f,c)` to signify that flight `f` starts from city `c`. In the ATIS database, however, this amounts to a path of three joins:

```
flight.from_airport-airport
     airport-airport_service
         airport_service-city
```

In GUSP, this is represented by the edge state `flight-flight.from_airport-airport-airport_service-city`.

GUSP only creates edge states for relational join paths up to length four, as longer paths rarely correspond to meaningful semantic relations.

Composition To handle compositions such as *American Airlines* and *New York City*, it helps to distinguish the head words (*Airlines* and *City*) from the rest. In GUSP, this is handled by introducing, for each node state such as `E:airline`, a new node state such as `E:airline:C`, where `C` signifies composition. For example, in Figure 1, *diego* is assigned to `V:city.name`, whereas *san* is assigned to `V:city.name:C`, since *san diego* forms a single meaning unit, and should be translated into SQL as a whole.

3.3 Domain-Independent States

These are for handling special linguistic phenomena that are not domain-specific, such as negation, superlatives, and quantifiers.

Operator states GUSP create node states for the logical and comparison operators (`OR`, `AND`, `NOT`, `MORE`, `LESS`, `EQ`). Additionally, to handle the cases when prepositions and logical connectives are collapsed into the label of a dependency edge, as in Stanford dependency, GUSP introduces an edge state for each triple of an operator and two node states, such as `E:flight-AND-E:fare`.

Quantifier states GUSP creates a node state for each of the standard SQL functions: `argmin, argmax, count, sum`. Additionally, it creates a node state for each pair of compatible function and property. For example, `argmin` can be applied to any numeric property, in particular `flight.departure_time`, and so the node state `P:flight.departure_time:argmin` is created and can be assigned to superlatives such as *earliest*.

3.4 Complex Semantic States

For sentences with a correct dependency tree and well-aligned syntax and semantics, the simple semantic states suffice for annotating the correct semantic parse. However, in complex sentences, syntax and semantic often diverge, either due to their differing goals or simply stemming from syntactic parsing errors. In Figure 1, the dependency tree contains multiple errors: *from toronto* and *to san diego* are mistakenly attached to *get*, which has no literal meaning here; *stopping in dtw* is also wrongly attached to *diego* rather than *flight*. Annotating such a tree with only simple states will lead to incorrect semantic parses, e.g., by joining `V:city:san diego` with `V:airport:dtw` via `E:airport_service`, rather than joining `E:flight` with `V:airport:dtw` via `E:flight_stop`.

To overcome these challenges, GUSP introduces three types of complex states to handle syntax-semantics divergence. Figure 1 shows the correct semantic parse for the above sentence using the complex states.

Raising For each simple node state `N`, GUSP creates a "raised" state `N:R` (`R` short for raised). A raised state signifies a word that has little or none of its own meaning, but effectively takes one of its child states to be its own ("raises"). Correspondingly, GUSP creates a "raising" edge state `N-R-N`, which signifies that the parent is a raised state and its meaning is derived from the dependency child of state `N`. For all other children, the parent behaves just as state `N`. For example, in Figure 1, *get* is assigned to the raised state `E:flight:R`, and the edge between *get* and *flight* is assigned to the raising edge state `E:flight-R-E:flight`.

Sinking For simple node states `A, B` and an edge state `E` connecting the two, GUSP creates a "sinking" node state `A+E+B:S` (`S` for sinking). When a node n is assigned to such a sinking state, n can behave as either `A` or `B` for its children (i.e., the edge states can connect to either one), and n's parent must be of state `B`. In Figure 1, for example, *diego* is assigned to a sinking state `V:city.name + E:flight` (the edge state is omitted for brevity). `E:flight` comes from its parent *get*. For child *san*, *diego* behaves as in state `V:city.name`, and their edge state is a simple compositional join. For the other child *stopping*, *diego* behaves as in state `E:flight`, and their edge state is a relational join connecting `flight` with `flight_stop`. Effectively, this connects *stopping* with *get* and eventually with *flight* (due to raising), virtually correcting the syntax-semantics mismatch stemming from attachment errors.

Implicit For simple node states `A, B` and an edge state `E` connecting the two, GUSP also creates a node state `A+E+B:I` (`I` for implicit) with the "implicit" state `B`. In natural languages, an entity is often introduced implicitly, which the reader infers from shared world knowledge. For example, to obtain the correct semantic parse for *Give me the fare from Seattle to Boston*, one needs to infer the existence of a *flight* entity, as in *Give me the fare (of a flight) from Seattle to Boston.* Implicit states offer candidates for addressing such needs. As in sinking, child nodes have access to either of the two simple states, but the implicit state is not visible to the parent node.

3.5 Lexical-Trigger Scores

GUSP uses the database elements to automatically derive a simple scoring scheme for lexical triggers. If a database element has a name of k words, each word is assigned score $1/k$ for the corresponding node state. Similarly for property values and value node states. In a sentence, if a word w triggers a node state with score s, its dependency children and left and right neighbors all get a trigger score of $0.1 \cdot s$ for the same state. To score relevant words not appearing in the database (due to incompleteness of the database or lexical variations), GUSP uses DASH (Pantel et al., 2009) to provide additional word-pair scoring based on lexical distributional similarity computed over general text corpora (Wikipedia in this case). In the case of multiple score assignments for the same word, the maximum score is used.

For multi-word values of property Y, and for a dependency edge connecting two collocated words, GUSP assigns a score 1.0 to the edge state joining the value node state `V:Y` to its composition state `V:Y:C`, as well as the edge state joining two composition states `V:Y:C`.

GUSP also uses a domain-independent list of superlatives with the corresponding data types and polarity (e.g., *first, last, earliest, latest, cheapest*) and assigns a trigger score of 1.0 for each property of a compatible data type (e.g., *cheapest* for properties of type `MONEY`).

3.6 The GUSP Model

In a nutshell, the GUSP model resembles a tree-HMM, which models the emission of words and dependencies by node and edge states, as well as transition between an edge state and the parent and child node states. In preliminary experiments on the development set, we found that the naïve model (with multinomials as conditional probabilities) did not perform well in EM. We thus chose to apply feature-rich EM (Berg-Kirkpatrick et al., 2010) in GUSP, which enabled the use of more generalizable features. Specifically, GUSP defines a probability distribution over dependency tree d and semantic parse z by

$$P_\theta(d, z) = \frac{1}{Z} \exp \sum_i f_i(d, z) \cdot w_i(d, z)$$

where f_i and w_i are features and their weights, and Z is the normalization constant that sums over all possible d, z (over the same unlabeled tree). The features of GUSP are as follows:

Lexical-trigger scores These are implemented as emission features with fixed weights. For example, given a token t that triggers node state N with score s, there is a corresponding features $1(lemma = t, state = \mathtt{N})$ with weight $\alpha \cdot s$, where α is a parameter.

Emission features for node states GUSP uses two templates for emission of node states: for raised states, $1(token = \cdot)$, i.e., the emission weights for all raised states are tied; for non-raised states, $1(lemma = \cdot, state = \mathtt{N})$.

Emission features for edge states GUSP uses the following templates for emission of edge states:

Child node state is `NULL`, dependency= ·;

Edge state is `RAISING`, dependency= ·;

Parent node state is same as the child node state, dependency= ·;

Otherwise, parent node state= ·, child node state= ·, edge state type= ·, dependency= ·.

Transition features GUSP uses the following templates for transition features, which are similar to the edge emission features except for the dependency label:

Child node state is `NULL`;

Edge state is `RAISING`;

Parent node state is same as the child node state;

Otherwise, parent node state= ·, child node state= ·, edge state type= ·.

Complexity Prior To favor simple semantic parses, GUSP imposes an exponential prior with weight β on nodes states that are not null or raised, and on each relational join in an edge state.

3.7 Learning and Inference

Since the GUSP model factors over nodes and edges, learning and inference can be done efficiently using EM and dynamic programming. Specifically, the MAP parse and expectations can be computed by tree-Viterbi and inside-outside (Petrov and Klein, 2008). The parameters can be estimated by feature-rich EM (Berg-Kirkpatrick et al., 2010).

Because the Viterbi and inside-outside are applied to a fixed tree (i.e., the input dependency tree), their running times are only linear in the sentence length in GUSP.

3.8 Query Generation

Given a semantic parse, GUSP generates the SQL by a depth-first traversal that recursively computes the denotation of a node from the denotations of its children and its node state and edge states. Each denotation is a structured query that contains: a list of entities for projection (corresponding to the `FROM` statement in SQL); a computation tree where the leaves are simple joins or value comparisons, and the internal nodes are logical or quantifier operators (the `WHERE` statement); the salient database elements (the `SELECT` statement). Below, we illustrate this procedure using the semantic parse in Figure 1 as a running example.

Value node state GUSP creates a semantic object of the given type with a unique index and the word constant. For example, the denotation for node *toronto* is a `city.name` object with a unique index and constant "toronto". The unique index is necessary in case the SQL involves multiple instances of the same entity. For example, the SQL in Figure 1 involves two instances of the entity `city`, corresponding to the departure and arrival cities, respectively. By default, such a semantic object will be translated into an equality constraint, such as $city.name = toronto$.

Entity or property node state GUSP creates a semantic object of the given type with a unique relation index. For example, the denotation for node *flight* is simply a `flight` object with a unique index. By default, such an object will contribute to the list of entities in SQL projection (the FROM statement), but not any constraints.

NULL state GUSP returns an empty denotation.

Simple edge state GUSP appends the child denotation to that of the parent, and appends equality constraints corresponding to the relational join path. In the case of composition, such as the join between *diego* and *san*, GUSP simply keeps the parent object, while adding to it the words from the child. In the case of a more complex join, such as that between *stopping* and *dtw*, GUSP adds the relational constraints that join `flight_stop` with `airport`:
$flight\_stop.stop\_airport = airport.airport\_id.$

Raising edge state GUSP simply takes the child denotation and sets that to the parent.

Implicit and sinking states GUSP maintains two separate denotations for the two simple states in the complex state, and processes their respective edge states accordingly. For example, the node *diego* contains two denotations, one for `V:city.name`, and one for `E:flight`, with the corresponding child being *san* and *stopping*, respectively.

Domain-independent states For comparator states such as `MORE` or `LESS`, GUSP changes the default equality constraints to an inequality one, such as $\texttt{flight.depart\_time} < 600$ for *before 6am*. For logical connectives, GUSP combines the projection and constraints accordingly. For quantifier states, GUSP applies the given function to the query.

Resolve scoping ambiguities GUSP delays applying quantifiers until the child semantic object differs from the parent one or when reaching the root. GUSP employs the following fixed ordering in evaluating quantifiers and operators: superlatives and other quantifiers are evaluated at last (i.e., after evaluating all other joins or operators for the given object), whereas negation is evaluated first, conjunctions and disjunctions are evaluated in their order of appearance.

4 Experiments

4.1 Task

We evaluated GUSP on the ATIS travel planning domain, which has been studied in He & Young (2005, 2006) and adapted for evaluating semantic parsing by Zettlemoyer & Collins (2007) (henceforth ZC07). The ZC07 dataset contains annotated logical forms for each sentence, which we do not use. Since our goal is not to produce a specific logical form, we directly evaluate on the end-to-end task of translating questions into database queries and measure question-answering accuracy. The ATIS distrbution contains the original SQL annotations, which we used to compute gold answers for evaluation only. The dataset is split into training, development, and test, containing 4500, 478, and 449 sentences, respectively. We used the development set for initial development and tuning hyperparameters. At test time, we ran GUSP over the test set to learn a semantic parser and output the MAP parses.[2]

4.2 Preprocessing

The ATIS sentences were originally derived from spoken dialog and were therefore in lower cases. Since case information is important for parsers and taggers, we first truecased the sentences using DASH (Pantel et al., 2009), which stores the case for each phrase in Wikipedia.

We then ran the sentences through SPLAT, a state-of-the-art NLP toolkit (Quirk et al., 2012), to conduct tokenization, part-of-speech tagging, and constituency parsing. Since SPLAT does not output dependency trees, we ran the Stanford parser over SPLAT parses to generate the dependency trees in Stanford dependency (de Marneffe et al., 2006).

4.3 Systems

For the GUSP system, we set the hyperparameters from initial experiments on the development set, and used them in all subsequent experiments. Specifically, we set $\alpha = 50$ and $\beta = -0.1$, and ran three iterations of feature-rich EM with an L_2 prior of 10 over the feature weights.

To evaluate the importance of complex states, we considered two versions of GUSP : **GUSP-SIMPLE** and **GUSP-FULL**, where **GUSP-SIMPLE** only admits simple states, whereas **GUSP-FULL** admits all states.

During development, we found that some questions are inherently ambiguous that cannot be solved except with some domain knowledge or labeled examples. In Section 3.2, we discuss an edge state that joins a flight with its starting city: `flight--flight.from_airport-airport--airport_service-city`. The ATIS database also contains another path of the same length: `flight-flight.from_airport--airport-ground_service-city`. The only difference is that `air_service` is replaced by `ground_service`. In some occasions, the answers are identical whereas in others they are different. Without other information, neither the complexity prior nor EM can properly discriminate one against another. (Note that this ambiguity is not present in the ZC07 logical forms, which use a single predicate `from(f,c)` for the entire relation paths. In other words, to translate ZC07 logical forms into SQL, one also needs to decide on which path to use.)

Table 1: Comparison of semantic parsing accuracy on the ATIS test dataset. Both ZC07 and FUBL used annotated logical forms in training, whereas GUSP-FULL and GUSP++ did not. The numbers for GUSP-FULL and GUSP++ are end-to-end question answering accuracy, whereas the numbers for ZC07 and FUBL are recall on exact match in logical forms.

	Accuracy
ZC07	84.6
FUBL	82.8
GUSP-FULL	74.8
GUSP++	83.5

Another type of domain-specific ambiguities involves sentences such as *give me information on flights after 4pm on wednesday*. There is no obvious information to disambiguate between `flight.departure_time` and `flight.arrival_time` for *4pm*.

Such ambiguities suggest opportunities for interactive learning,[3] but this is clearly out of the scope of this paper. Instead, we incorporated a simple disambiguation feature with a small weight of 0.01 that fires over the simple states of `flight.departure_time` and `airport_service`. We named the resulting system **GUSP**++.

To gauge the difficulty of the task and the quality of lexical-trigger scores, we also considered a deterministic baseline **LEXICAL**, which computed semantic parses using lexical-trigger scores alone.

[2] This doesn't lead to overfitting since we did not use any labeled information in the test set.

[3] For example, after eliminating other much less likely alternatives, the system can present to the user with both choices and let the user to choose the correct one. The implicit feedback signal can then be used to train the system for future disambiguation.

Table 2: Comparison of question answering accuracy in ablation experiments.

	Accuracy
LEXICAL	33.9
GUSP-SIMPLE	66.5
GUSP-FULL	74.8
GUSP++	83.5
– RAISING	75.7
– SINKING	77.5
– IMPLICIT	76.2

4.4 Results

We first compared the results of GUSP-FULL and GUSP++ with ZC07 and FUBL (Kwiatkowski et al., 2011).[4] Note that ZC07 and FUBL were evaluated on exact match in logical forms. We used their recall numbers which are the percentages of sentences with fully correct logical forms. Given that the questions are quite specific and generally admit nonzero number of answers, the question-answer accuracy should be quite comparable with these numbers.

Table 1 shows the comparison. Surprisingly, even without the additional disambiguation feature, GUSP-FULL already attained an accuracy broadly in range with supervised results. With the feature, GUSP++ effectively tied with the best supervised approach.

To evaluate the importance of various components in GUSP, we conducted ablation test to compare the variants of GUSP. Table 2 shows the results. LEXICAL can parse more than one third of the sentences correctly, which is quite remarkable in itself, considering that it only used the lexical scores. On the other hand, roughly two-third of the sentences cannot be correctly parsed in this way, suggesting that the lexical scores are noisy and ambiguous. In comparison, all GUSP variants achieved significant gains over LEXICAL. Additionally, GUSP-FULL substantially outperformed GUSP-SIMPLE, highlighting the challenges of syntax-semantics mismatch in ATIS, and demonstrating the importance and effectiveness of complex states for handling such mismatch. All three types of complex states produced significant contributions. For example, compared to GUSP++, removing RAISING dropped accuracy by almost 8 points.

4.5 Discussion

Upon manual inspection, many of the remaining errors are due to syntactic parsing errors that are too severe to fix. This is partly due to the fact that ATIS sentences are out of domain compared to the newswired text on which the syntactic parsers were trained. For example, *show*, *list* were regularly parsed as nouns, whereas *round* (as in *round trip*) were often parsed as a verb and *northwest* were parsed as an auxilliary verb. Another reason is that ATIS sentences are typically less formal or grammatical, which exacerbates the difficulty in parsing. In this paper, we used the 1-best dependency tree to produce semantic parse. An interesting future direction is to consider joint syntactic-semantic parsing, using k-best trees or even the parse forest as input and reranking the top parse using semantic information.[5]

5 Conclusion

This paper introduces grounded unsupervised semantic parsing, which leverages available database for indirect supervision and uses a grounded meaning representation to account for syntax-semantics mismatch in dependency-based semantic parsing. The resulting GUSP system is the first unsupervised approach to attain an accuracy comparable to the best supervised systems in translating complex natural-language questions to database queries.

Directions for future work include: joint syntactic-semantic parsing, developing better features for learning; interactive learning in a dialog setting; generalizing distant supervision; application to knowledge extraction from database-rich domains such as biomedical sciences.

Acknowledgments

We would like to thank Kristina Toutanova, Chris Quirk, Luke Zettlemoyer, and Yoav Artzi for useful discussions, and Patrick Pantel and Michael Gammon for help with the datasets.

[4] We should note that while the more recent system of FUBL slightly trails ZC07, it is language-independent and can parse questions in multiple languages.

[5] Note that this is still different from the currently predominant approaches in semantic parsing, which learn to parse both syntax and semantics by training from the *semantic parsing datasets* alone, which are considerably smaller compared to resources available for syntactic parsing.

References

Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. 2007. Open information extraction from the web. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 2670–2676, Hyderabad, India. AAAI Press.

Taylor Berg-Kirkpatrick, John DeNero, and Dan Klein. 2010. Painless unsupervised learning with features. In *Proceedings of Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

Benjamin Börschinger, Bevan K. Jones, and Mark Johnson. 2011. Reducing grounded learning tasks to grammatical inference. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*.

Xavier Carreras and Luis Marquez. 2004. Introduction to the CoNLL-2004 shared task: Semantic role labeling. In *Proceedings of the Eighth Conference on Computational Natural Language Learning*, pages 89–97, Boston, MA. ACL.

David L. Chen and Raymond J. Mooney. 2008. Learning to sportscast: A test of grounded language acquisition. In *ICML-08*.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from world's response. In *Proceedings of the 2010 Conference on Natural Language Learning*.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation*, pages 449–454, Genoa, Italy. ELRA.

Pedro Domingos and Daniel Lowd. 2009. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool, San Rafael, CA.

Dan Goldwasser, Roi Reichart, James Clarke, and Dan Roth. 2011. Confidence driven unsupervised semantic parsing. In *Proceedings of the Forty Ninth Annual Meeting of the Association for Computational Linguistics*.

B.J. Grosz, D. Appelt, P. Martin, and F. Pereira. 1987. Team: An experiment in the design of transportable natural language interfaces. *Artificial Intelligence*, 32:173–243.

Yulan He and Steve Young. 2005. Semantic processing using the hidden vector state model. In *Computer Speech and Language*.

Yulan He and Steve Young. 2006. Spoken language understanding using the hidden vector state model. In *Speech Communication Special Issue on Spoken Language understanding for Conversational Systems*.

Larry Heck, Dilek Hakkani-Tur, and Gokhan Tur. 2013. Leveraging knowledge graphs for web-scale unsupervised semantic parsing. In *Proceedings of the Interspeech 2013*.

Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S. Weld. 2011. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the Forty Ninth Annual Meeting of the Association for Computational Linguistics*.

R. J. Kate, Y. W. Wong, and R. J. Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*.

Joohyun Kim and Raymond J. Mooney. 2010. Generative alignment and semantic parsing for learning from ambiguous supervision. In *COLING10*.

Jayant Krishnamurthy and Tom M. Mitchell. 2012. Weakly supervised training of semantic parsers. In *EMNLP-12*.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in ccg grammar induction for semantic parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*.

Percy Liang, Michael Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of the Forty Ninth Annual Meeting of the Association for Computational Linguistics*.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Proceedings of the Forty Seventh Annual Meeting of the Association for Computational Linguistics*.

Raymond J. Mooney. 2007. Learning for semantic parsing. In *Proceedings of the Eighth International Conference on Computational Linguistics and Intelligent Text Processing*, pages 311–324, Mexico City, Mexico. Springer.

Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. 2009. Web-scale distributional similarity and entity set expansion. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*.

Slav Petrov and Dan Klein. 2008. Discriminative log-linear grammars with latent variables. In *NIPS-08*.

Hoifung Poon and Pedro Domingos. 2007. Joint inference in information extraction. In *Proceedings of the Twenty Second National Conference on Artificial Intelligence*, pages 913–918, Vancouver, Canada. AAAI Press.

Hoifung Poon and Pedro Domingos. 2009. Unsupervised semantic parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1–10, Singapore. ACL.

Hoifung Poon and Pedro Domingos. 2010. Unsupervised ontological induction from text. In *Proceedings of the Forty Eighth Annual Meeting of the Association for Computational Linguistics*, pages 296–305, Uppsala, Sweden. ACL.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *IUI-03*.

Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *COLING-04*.

Chris Quirk, Pallavi Choudhury, Jianfeng Gao, Hisami Suzuki, Kristina Toutanova, Michael Gamon, Wentau Yih, and Lucy Vanderwende. 2012. MSR SPLAT, a language analysis toolkit. In *Proceedings of NAACL HLT 2012 Demonstration Session*.

Sebastian Riedel, Limin Yao, and Andrew McCallum. 2010. Modeling relations and their mentions without labeled text. In *Proceedings of the Sixteen European Conference on Machine Learning*.

Ivan Titov and Alexandre Klementiev. 2011. A bayesian model for unsupervised semantic parsing. In *Proceedings of the Forty Ninth Annual Meeting of the Association for Computational Linguistics*.

Ivan Titov and Alexandre Klementiev. 2012. A bayesian approach to unsupervised semantic role induction. In *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammers. In *Proceedings of the Twenty First Conference on Uncertainty in Artificial Intelligence*, pages 658–666, Edinburgh, Scotland. AUAI Press.

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