Introduction to Semantic Parsing

Kilian Evang

June 7, 2018

Contents

List of Abbreviations

NL natural language (e.g., English or German)

NLU natural language utterance (e.g., a sentence, a question, or a command)

MRL meaning representation language, a formal language (e.g., the lambda calculus or SQL)

MR meaning representation (e.g. a first-order logic formula or an SQL query)

1 Introduction

A semantic parser is a program that automatically translates natural-language utterances (NLUs) to formal meaning representations (MRs) that a computer can execute. For example, the NLU in (1-a) is a question that might be posed to a geographical information system. A semantic parser could translate it into an MR such as (1-b), which can just be run against a suitable (Prolog) database to produce the answer.

(1) a. Give me the cities in Virginia.

b. `answer(A, (city(A), loc(A, B), const(B, stateid(virginia))))`

A traditional way to create a semantic parser is to write a program that transforms NLUs (or syntactic parse trees of them) into MRs according to rules. However, writing such rules is complicated and requires special skills. And even if the subject domain of the desired parser is quite limited (e.g., to U.S. geography), many rules will be needed. This is because of the great variability of natural language. Every MR usually has many ways it could be phrased in natural language. For example, (2-a–c) should all map to (2-d):

(2)
a. What's the population of Alabama?
b. How many people live in Alabama?
c. How many citizens in Alabama?
d. `answer(A, (population(B, A), const(B, stateid(alabama))))`

We therefore want computers to learn the rules needed to transform NLUs into MRs by themselves—from examples such as those shown in (1) and (2). This is also referred to as *semantic parser learning*. This is the kind of semantic parsing we will explore in this seminar.

2 Characteristics of a Semantic Parser

Semantic parsers differ in their target applications and techniques. Specifically, they differ in the following respects:

1. NLU representation—the form in which NLUs are fed into the parser
2. MRL—the formal language that the parser outputs
3. data—the amount and level of detail of training data that the parser needs
4. lexicon representation—in what format the parser associates words (and multiword expressions) with partial MRs
5. candidate lexicon generation—how the parser goes about creating such an association
6. parsing algorithm—how the parser goes about transforming an NLU into an MR
7. features—what kinds of features the parser uses to guide the parsing algorithm towards the correct solution
8. model—how the parser uses the features to determine the correct decisions
9. learning algorithm—how the parser learns the model and refines the candidate lexicon into a more accurate lexicon
10. experimental setup—how the parser is trained and tested (not strictly part of the parser itself, but of the research for creating it)

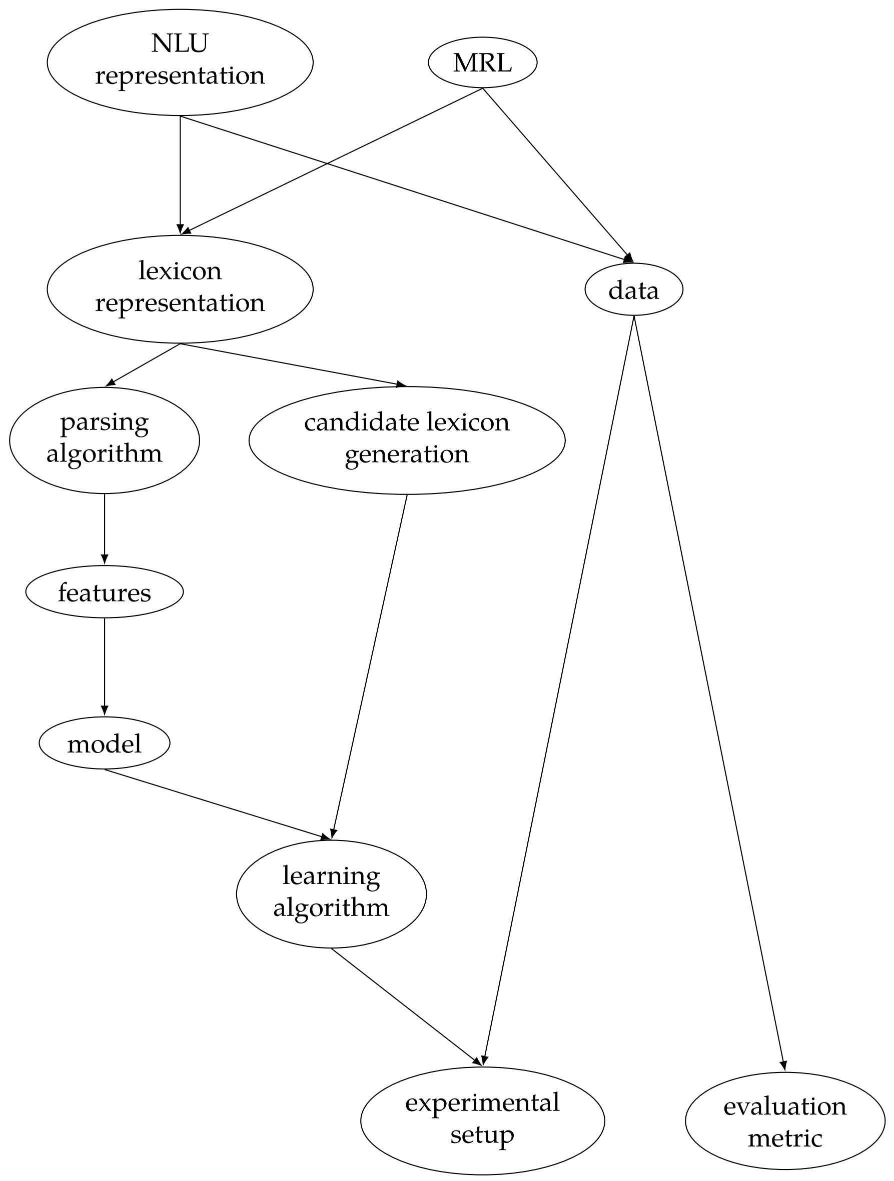


Figure 1: Characteristics of a semantic parser. An arrow pointing from A to B means that A should be understood first to understand B.

11. evaluation metric—how the output of the parser is scored for correctness (not strictly part of the parser itself, but of the research for creating it)

Figure 1 shows how these topics build on each other. In the following sections, we will discuss them one by one. As an example, we will introduce a simple semantic parser called GEOPAR[1], which deals with geographical queries as shown in (1) and (2). For each topic, GEOPAR will take one of the simplest possible approaches. We will also point to other possible approaches in each section.

2.1 NLU Representation

GEOPAR will take queries as input simply as lists of lower-cased tokens. For example (in Prolog format):

(3)
```
[what,is,the,capital,of,the,state,with,the,largest,
population,?]
```

Other Possibilities

Instead of just giving the parser the tokens, one might pre-process them to add part-of-speech tags and/or word vectors (also called word embeddings). With part-of-speech tags, the parser would know, e.g., that `of` and `in` are both prepositions, and could more easily learn that they have similar properties. Similarly, word vectors allow the parser to recognize similarities between words (such as `in` and `of`, `king` and `queen`), but they capture more similarities and more fine-grained similarities.[2]

2.2 MRL

GEOPAR will produce queries in the GEOQUERY query language, introduced by Zelle and Mooney (1996). The GEOQUERY representation of the question in (3) is:

(4)
```
answer(C, (capital(C), loc(C, S), largest(P, (state(S),
population(S, P)))))
```

Every query has the form `answer(_, _)`. The first argument is a variable that stands for the answer that a geographical information system asked that query would be expected to give. The second argument consists of constraints imposed on that entity: it should be a capital (`capital(C)`), it should be located in `S` (`loc(C, S)`), and that `S` should be the state with the largest population (`largest(P, (state(S), population(S, P)))`). For further details on GEOQUERY, see Zelle and Mooney (1996), pp. 1052 ff.

[1]Your task during the practical sessions of this course will be to implement and then extend GEOPAR.

[2]One good Python library for assigning part-of-speech tags and word vectors to tokens (among other things) is spaCy (`https://spacy.io`).

Other Possibilities

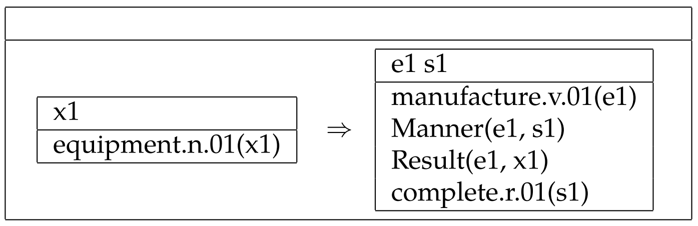
Restricted to factoid questions about United States geography, GeoQuery is a simple MRL that was primarily used as a benchmark in the early years of semantic parsing research. Similar MRLs include Atis for queries on flight information (Price, 1990; Bates et al., 1990) and Clang for instructions to robotic soccer players (Kate et al., 2005). Here is an example of an NLU paired with an Atis MR, adapted from Papineni et al. (1997):

(5) a. What are the least expensive flights from Baltimore to Seattle?
b. LIST FLIGHTS CHEAPEST FROM:CITY BALTIMORE TO:CITY SEATTLE

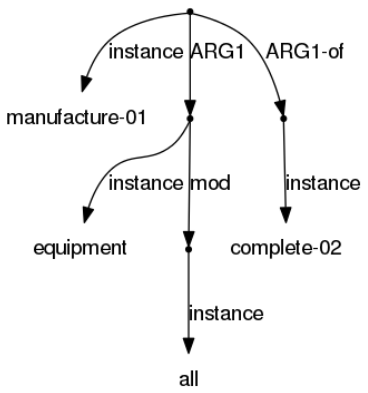
As the state of the art progressed, semantic parsers were developed that used query languages with much larger vocabularies, suitable for querying large, general knowledge bases such as Freebase (Krishnamurthy and Mitchell, 2012). Most recently, semantic parsers are being developed that produce complex meaning representations intended to capture the meaning of arbitary sentences and texts. Examples of such MRLs are Discourse Representation Structures (Kamp, 1984; Kamp and Reyle, 1993; Bos et al., 2017; Abzianidze et al., 2017) and Abstract Meaning Representations (Banarescu et al., 2013). (6-a) shows an example NLU, (6-b) shows a corresponding DRS, and (6-c) shows a corresponding AMR (adapted from Bjerva et al. (2016):

(6) a. All equipment will be completely manufactured.

b.



c.



2.3 Data

Semantic parsers learn how to translate NLUs into MRs. They do this by looking at examples, called the *training data*. In many cases, this data consists simply of example NLUs paired with the correct resulting MR (manually created). For example:

(7)
```
parse([what,is,the,capital,of,the,state,with,the,largest,
population,?], answer(A,(capital(A),loc(A,B),largest(C,(
state(B),population(B,C))))).
```

Other Possibilities

Producing an MR is not the end goal for any semantic parser. Oftentimes, the MR is just a query that is sent to a database or other information system to find the answer to the question. It therefore makes sense to say "I don't care what MR the parser produces, so long as the database gives back the correct answer when fed the MR." (The correct answer, in our example, would be `cityid(sacramento, ca)`.)

Accordingly, some semantic parsers do not require gold-standard meaning representations in the training data. They are just given the correct answers and access to the database and learn to produce meaning representations that yield the desired answers. This approach is also called *weakly supervised semantic parsing* and was pioneered by Clarke et al. (2010) and Liang et al. (2011). Its most important advantage is that it is much easier to create training data this way, since it can be done by people without specialized knowledge about MRs. A disadvantage is that learning becomes more difficult, because the parser has less information to go by and so has to try out many different possibilities. Nevertheless, weakly supervised learning has turned out to work very well for small domains such as GeoQuery.

There are approaches to learning semantic parsers with even less explicit training data, where only the database and example NLUs are given, with no answers. This is called *unsupervised semantic parsing* (Poon and Domingos, 2009; Goldwasser et al., 2011). It must be combined with advanced methods for candidate lexicon generation (see below) to be effective.

Yet another form of supervision is *cross-lingual supervision*, applicable when a semantic parser for, say, English already exists and a new semantic parser for, say, Dutch should be learned. This approach also assumes the existence of a parallel corpus (i.e., sentences paired with their translations). The training data for the new parser is then generated using the existing parser (Evang and Bos, 2016).

2.4 Evaluation Metric

An evaluation metric for semantic parsing is a function that compares a gold MR with the MR produced by a semantic parser and outputs some number between 0 and 1 saying how close the parser came to being correct. When averaged over a representative collection of NLU-MR pairs, an evaluation metric can be used to compare the performance of semantic parsers, and to check whether any particular change made to a semantic parser during development hurt or improved its performance.

The simplest evaluation metric, called *exact match*, simply returns 1 if the MRs are the same, and 0 otherwise. This is the metric we will initially be using to evaluate Geopar.

Other Possibilities

The exact match metric is sometimes problematic, because some differences don't matter, such as the order of subformulas in conjunctions. For example, the following two MRs are logically equivalent:

(8) a. `answer(C, (capital(C), loc(C, S), largest(P, (state(S), population(S, P)))))`
b. `answer(C, (loc(C, S), capital(C), largest(P, (state(S), population(S, P)))))`

Thus, evaluation metrics are sometimes designed to ignore such differences.

Another possibility is *in-vivo evaluation*. Similarly to the argument made above for training data, we may not care about the MR itself, but about the result when feeding it into an information system. Thus, semantic parsers can also be evaluated purely by whether the correct answer results.

For more complex meaning representations, parsers are usually assigned partial credit if they don't get the whole MR right but some of its substructures. An example of such a metric is the Smatch metric for Abstract Meaning Representations (Cai and Knight, 2013).

2.5 Lexicon Representation

Words. They mean things.

The Linguist Llama

One of the most important things a semantic parser needs to learn is what individual words mean (*lexicon learning*). A word, after all, can make all the difference in meaning to an NLU—for example:

(9) a. Give me the cities in Virginia.
b. `answer(A,(city(A),loc(A,B),const(B,stateid(virginia))))`

(10) a. Give me the rivers in Virginia.
b. `answer(A,(river(A),loc(A,B),const(B,stateid(virginia))))`

One of the design decisions when creating a semantic parser is how to represent the meanings of individual words, called *lexical MRs* or *lexical semantics*. The possible choices depend on the MRL. For Geopar, we define the possible lexical MRs to be

1. complex terms with functor name `const` and two arguments, the first being a variable and the second a term that represents an individual entity. For example:
   - `const(_,stateid(virginia))`
   - `const(_,riverid(mississippi))`
   - `const(_,cityid(austin, texas))`
2. complex terms with other functor names where all arguments are variables. For example:

- `city(_)`
- `loc(_,_)`
- `river(_)`
- `largest(_,_)`

As in Prolog, we use the underscore to represent variables that only occur once. For example, in `loc(_,_)`, the two variables are different. It will be the task of the parsing algorithm to assemble lexical MRs into full MRs and to create the appropriate variable bindings in the process.

A lexicon is then a set of *lexicon entries* $W \vdash M$ where W is a list of one or more NL words and M is a lexical MR. For example, a minimal lexicon to parse the examples (9) and (10) could look like this:[3]

(11) { `[cities]`⊢`city(_)`, `[rivers]`⊢`river(_)`, `[in]`⊢`loc(_,_)`, `[virginia]`⊢`const(_,stateid(virginia))` }

Sometimes it makes sense to associate more than one NL word with a lexical MR—for example, when several NL words are used to express a concept that has a single symbol in the MRL, or in the case of names that consist of more than one word. Consider the following example:

(12) a. How big is New Mexico?
b. `answer(A,(size(B,A),const(B,stateid('new mexico'))))`

Here are some lexical entries that a semantic parser might sensibly learn. Some of them are multiword lexical entries:

(13) a. `[big]`⊢`size(_,_)`
b. `[how,big]`⊢`size(_,_)`
c. `[how,big,is]`⊢`size(_,_)`
d. `[new,mexico]`⊢`const(_,stateid('new mexico'))`

Other Possibilities

Many semantic parsers use the lambda calculus to represent both full MRs and lexical MRs. Like our lexical MRs above, lambda terms have variables where terms coming from other lexical entries can be "filled in". Unlike our lexical MRs above, lambda terms have prefixes that specify explicitly which variables these are and in which order they should be filled in.

Here is an example NLU-MR pair from Zettlemoyer and Collins (2005):

(14) a. What states border Texas?
b. $\lambda x.(state(x) \land borders(x, texas))$

Their semantic parser uses the following lexical entries to parse this example:

(15) a. What $\vdash \lambda f.\lambda g.\lambda x.(f(x) \land g(x))$
b. states $\vdash \lambda x.state(x)$
c. border $\vdash \lambda x.\lambda y.borders(y, x)$
d. Texas $\vdash texas$

[3]Every GeoQuery MR is wrapped in `answer(_,_)`, so we don't treat this as a lexical MR.

A lambda term $\lambda x.P$ can be seen as a function with a parameter x which, when given an argument, returns P with x replaced by that argument. Thus, when we apply the lexical MRs above to each other in the right order, we get the full MR:

$$\begin{aligned}
\lambda f.\lambda g.\lambda x.(f(x) \land g(x))(\lambda x.\mathit{state}(x)) &= \lambda g.\lambda x.(\mathit{state}(x) \land g(x)) \\
\lambda x.\lambda y.\mathit{borders}(y, x)(\mathit{texas}) &= \lambda y.\mathit{borders}(y, \mathit{texas}) \\
\lambda g.\lambda x.(\mathit{state}(x) \land g(x))(\lambda y.\mathit{borders}(y, \mathit{texas})) &= \lambda x.(\mathit{state}(x) \land \mathit{borders}(x, \mathit{texas}))
\end{aligned}$$

2.6 Candidate Lexicon Generation

As already hinted at above, a semantic parser has two main learning tasks:

1. Lexicon learning: how to map words to lexical MRs.
2. Parser learning: how to assemble the lexical MRs of an NLU into a full MR.

These two tasks are intertwined, because you can't parse unless you have a lexicon, and you don't know which lexical entries are correct unless you use them for parsing, at which point you see whether you end up with the correct full MR.

A common strategy to overcome this catch-22 is to first generate a *candidate lexicon* which hopefully contains all correct lexical entries, but also many incorrect ones. The parser then first uses this candidate lexicon to parse. By comparing the results to the gold standard MRs in the training data, it can learn not only how to assemble MRs, but also which lexical entries to use at all.

Since the vocabulary of GeoQuery is very small, to simplify things, we will first equip Geopar with a hand-written lexicon. Later in the course, we will consider automatic approaches to candidate lexicon generation.

Other Possibilities

Most approaches to candidate lexicon generation exploit co-occurrence information. Consider this small corpus:

(16) a. What is the smallest state?
b. `answer(S,(smallest(S,state(S)))`

(17) a. What is the largest state?
b. `answer(S,(largest(S,state(S)))`

Looking at just (16), a semantic parser could not know that *smallest* means `smallest(_,_)` and *state* means `state(_)` and not the other way round. But looking at both (16) and (17), it can notice that the NL word *state* appears in both NLUs, and the lexical MR `state(_)` appears in both MRs. This is a clue that they should be associated. Many semantic parsers use *unsupervised word alignment* techniques originally developed for machine translation to figure out correlations and likely associations between words and lexical MRs, and generate a candidate lexicon from that.

The candidate lexical MRs are extracted from the MRs in the training data. For GEOPAR, we assume that lexical MRs always follow one of the two forms specified above, so they are easy enough to extract from the full MRs. Parsers that use the lambda calculus have a bit more work to do. There are two main approaches here: lexical lambda-terms can be extracted using hand-written templates (Zettlemoyer and Collins, 2005, 2007) or found by recursively splitting the full MR into smaller parts (Kwiatkowksi et al., 2010; Kwiatkowski et al., 2011). In the case of cross-lingual semantic parsing, lexical MRs are transferred from the source-language lexical entries to the target-language words (Evang and Bos, 2016).

For scaling semantic parsers to vocabularies larger than GEOQUERY, there is rarely enough training data to learn all lexical entries directly from it. Instead, there is much work on filling gaps in the lexicon by looking at databases and unannotated text (Krishnamurthy and Mitchell, 2012; Cai and Yates, 2013b,a; Berant et al., 2013; Kwiatkowski et al., 2013; Reddy et al., 2014; Wang et al., 2015).

2.7 Parsing Algorithm

Once a (candidate) lexicon exists, a parsing algorithm is needed to get from a list of words to a full MR. GEOPAR, like many parsers today, will use a transition-based (also called shift-reduce) parsing algorithm. Transition-based algorithms work "from left to right", processing the words of the NLU in order and integrating the lexical MRs of new words into the whole MR as soon as possible.

The parser uses a special data structure, called *parse item*. A parse item is a triple $\langle S, Q, F \rangle$. S is called the *stack* and Q is called the *queue*. F is 0 or 1, indicating whether the item is "finished". The queue initially contains the whole NLU to parse. The parsing process consists in removing words from the queue one by one, putting their lexical MRs onto the stack and building up the full MR on the stack, step by step.

Parsing starts with the *initial item*. The initial item has a stack with one element (`answer(_,_)`) and the whole NLU in the queue. The parser then executes a series of actions until it arrives at a finished item, with the queue empty and a single element on the stack, which is the final MR. There are seven kinds of actions, given a lexicon L:

- SHIFT: remove the first N words from the queue, find a lexicon entry for these words and put the lexical MR on top of the stack.
- SKIP: remove the first word from the queue. Used primarily for words that don't have a lexicon entry, e.g., articles.
- COREF: create a variable binding between the top two elements of the stack.
- DROP: embed the topmost stack element into the second-topmost one.
- LIFT: embed the second-topmost stack element into the topmost one.
- FINISH: only applies to items with an empty queue and a single element on the stack. Change F from 0 to 1, marking the item as finished.
- IDLE: only applies to finished items. Does not change anything.

Action	Parse item
	⟨⟨answer(_, _)⟩, ⟨what, is, the, capital, of, the, state, with, the, largest, population⟩, 0⟩
SKIP	⟨⟨answer(_, _)⟩, ⟨is, the, capital, of, the, state, with, the, largest, population⟩, 0⟩
SKIP	⟨⟨answer(_, _)⟩, ⟨the, capital, of, the, state, with, the, largest, population⟩, 0⟩
SKIP	⟨⟨answer(_, _)⟩, ⟨capital, of, the, state, with, the, largest, population⟩, 0⟩
SHIFT-1-capital(_, _)	⟨⟨answer(_, _), capital(_, _)⟩, ⟨of, the, state, with, the, largest, population⟩, 0⟩
COREF-[1]-[2]	⟨⟨answer(C, _), capital(_, C)⟩, ⟨of, the, state, with, the, largest, population⟩, 0⟩
DROP-[2]	⟨⟨answer(C,capital(_, C))⟩, ⟨of, the, state, with, the, largest, population⟩, 0⟩
SKIP	⟨⟨answer(C,capital(_, C))⟩, ⟨the, state, with, the, largest, population⟩, 0⟩
SKIP	⟨⟨answer(C,capital(_, C))⟩, ⟨state, with, the, largest, population⟩, 0⟩
SHIFT-1-state(_)	⟨⟨answer(C,capital(_, C)), state(_)⟩, ⟨with, the, largest, population⟩, 0⟩
COREF-[2, 1]-[1]	⟨⟨answer(C,capital(S, C)), state(S)⟩, ⟨with, the, largest, population⟩, 0⟩
SKIP	⟨⟨answer(C,capital(S, C)), state(S)⟩, ⟨the, largest, population⟩, 0⟩
SKIP	⟨⟨answer(C,capital(S, C)), state(S)⟩, ⟨largest, population⟩, 0⟩
SHIFT-1-largest(_, _)	⟨⟨answer(C,capital(S, C)), state(S), largest(_, _)⟩, ⟨population⟩, 0⟩
LIFT-[2]	⟨⟨answer(C,capital(S, C)), largest(_,state(S))⟩, ⟨population⟩, 0⟩
SHIFT-1-population(_, _)	⟨⟨answer(C,capital(S, C)), largest(_,state(S)), population(_, _)⟩, ⟨⟩, 0⟩
COREF-[2, 1]-[1]	⟨⟨answer(C,capital(S, C)), largest(_,state(S)), population(S, _)⟩, ⟨⟩, 0⟩
COREF-[1]-[2]	⟨⟨answer(C,capital(S, C)), largest(P,state(S)), population(S, P)⟩, ⟨⟩, 0⟩
DROP-[2]	⟨⟨answer(C,capital(S, C)), largest(P, (state(S),population(S, P)))⟩, ⟨⟩, 0⟩
DROP-[2]	⟨⟨answer(C, (capital(S, C),largest(P, (state(S), population(S, P)))))⟩, ⟨⟩, 0⟩
FINISH	⟨⟨answer(C, (capital(S, C),largest(P, (state(S), population(S, P)))))⟩, ⟨⟩, 1⟩
IDLE	⟨⟨answer(C, (capital(S, C),largest(P, (state(S), population(S, P)))))⟩, ⟨⟩, 1⟩
...	

Table 1: An example parsing process.

COREF, DROP, and LIFT can operate not only on the two topmost terms, but also on subterms. For this purpose, they have lists of argument numbers as arguments. For example, COREF-[2, 1]-[1] means: create a variable binding between the first argument of the second argument of the second-topmost stack element, and the first argument of the topmost stack element. And DROP-[2] means: drop the topmost stack element into the second argument o the second-topmost stack element.

Table 1 shows the parsing process for one sentence.

When parsing new NLUs (outside the training data), GEOPAR will not necessarily know which action is the right action to take at each step. For example, when the word `mississippi` is at the front of the queue, should the action SHIFT-`const(_, stateid(mississippi))` or SHIFT-`const(_, riverid(mississippi))` be used? To allow for some (temporary) error, we will use *beam search*. That is, at each step, the parser will apply every possible action, resulting in many alternative new parse items. Of these new parse items, only the (say) 10 with the highest scores are kept. They make up the *beam*. In the next step, every possible action is applied to every item in the beam. Many new items result, and again only the 10 with the highest scores are kept, making up the beam for the next step. And so on, until all items in the beam are finished. (This is the reason why we have IDLE: so that finished items can be kept in the beam while others are still unfinished.)

What are "scores" of parse items? We will see in the next section.

Other Possibilities

Instead of a transition-based algorithm, some semantic parsers use a chart-based algorithm such as CKY. Instead of from left to right, chart-based algorithms work from bottom to top, building MRs for smaller parts of the NLU first, then for larger parts, and finally for the whole NLU. Transition-based algorithms tend to be favored today because they work in linear time, are just as performant, arguably conceptually simpler and more flexible, and they are more similar to how the human brain (the best known semantic parser) processes sentences: starting with the first word and starting to interpret stuff even before the sentence is fully spoken.

2.8 Features and Model

Because our lexicon and parsing actions typically allow for more than one possible action at each step, the parser needs some way of comparing different parse items to each other and determining which is the most likely to be correct. We do this by assigning each parse item a *score*, which is simply a number—the higher it is, the more convinced the parser is the item is correct. The score is computed by looking at a number of binary *features* of the parse item. Each feature is a simple statement about a parse item which is either true or false.

Some features that are true of the fourth parse item in Table 1 are:

(18)
a. The first word in the queue is `capital`.
b. The second word in the queue is `of`.
c. The last action was `skip`.
d. The last processed word was `the`.
e. The second-to-last processed word was `is`.
f. The predicate at $\boxed{0}$ is `answer/2`.
g. The predicate at $\boxed{1}$ is `none`.

Some features that are true of the ninth parse item are:

(19)
a. The first word in the queue is `state`.
b. The second word in the queue is `with`.
c. The last action was `skip`.
d. The last processed word was `the`.
e. The second-to-last processed word was `of`.
f. The predicate at $\boxed{0}$ is `answer/2`.
g. The predicate at $\boxed{1}$ is `capital/2`.

Note that (18-a) is a positive clue that the action SHIFT-`capital(_, _)` should come next, but a negative clue that the action SHIFT-`state(_, _)` should not come next. With (19-a), it is the other way around. Features (18-d) and (19-d) are the same for both items, so they are not a particularly strong clue for one vs. the other. It may however be a negative clue against a SKIP action, because *the* is typically followed by a noun, and nouns typically introduce predicates.

Parser learning (see below) is figuring out which features are strong/weak positive/negative indicators of which actions, and assigning feature-action pairs $\langle f, a \rangle$ high/low positive/negative *weights* accordingly. The model provides a *weight function* ω such that $\omega(\langle f, a \rangle)$ returns a number. An item j is then scored based on the features that are true of its predecessor i and the action a that was taken to get from i to j. Specifically, the *local score* of j is $\sum_{f \text{ is true of } i} \omega(\langle f, a \rangle)$. The *global score* of j is the sum of the local scores of all items leading up to it, including j itself. And the global score is what is used to determine which items may stay in the beam, and to select the final winning MR at the end.

Finally, a word about how we find the features that are true for a given item: we use *feature templates*. For the example features in (18) and (19), we used the following templates:

(20) a. The first word in the queue is _.
b. The second word in the queue is _.
c. The last action was _.
d. The last processed word was _.
e. The second-to-last processed word was _.
f. The predicate at $\boxed{0}$ is _.
g. The predicate at $\boxed{1}$ is _.

Other Possibilities

What we have defined above is a simple global linear model with sparse features, meaning that there are many possible features, only few of which are usually true of any given item. This approach requires careful definition and testing of many feature templates to work well. One way to improve it both in terms of manual labor required and in terms of performance is to use a *neural network* with *dense features*. Instead of manually defining feature templates, here, we represent words as word vectors (see above) and use a variety of techniques such as convolutional layers, recurrent layers and attention mechanisms to allow the learning algorithm to figure out the relevant features itself. Prediction of the right action at each step would then be made using a softmax layer.

2.9 Training Algorithm

Initially, we don't know what good weights to use are—we need to train the model first. For Geopar, we will do this by parsing the training data and changing the weights a bit each time the parser doesn't arrive at the correct MR, pushing it towards making the correct decisions step by step. A sketch of the training algorithm is given in Algorithm 1.

Other Possibilites

Depending on the model used (e.g., linear vs. log-linear vs. neural), training algorithms differ somewhat. One major topic we have not addressed so far is lexicon learning, which in many semantic parsers is interleaved with parser learning: starting out with an initial candidate lexicon, the parser alternates between training the weights, introducing new lexical entries and discarding lexical entries that do not seem useful.

2.10 Experimental Setup

As for all machine learning experiments, the ultimate goal is to have the system come up with the correct answers for completely new data so it can be applied to tasks in the real world. To simulate "new" data, we set a part of the data we have aside as *test data*. This is the data we will use at the very end to test our semantic parser on and to assess how it would perform on new data. We must not train our system on the test data, and we should not prematurely evaluate it on the test data and then make changes to it based on the results. This is because we cannot do that with unseen real-world data either.

Algorithm 1 A sketch of the training algorithm.

```
Input: a number of training NLU-MR pairs $\langle x, y\rangle$, a lexicon $L$, a beam size $b$
Output: a weight function $\omega$
Initialize $\omega$ to be 0 for every feature-action combination $\langle f, a\rangle$
for a number of epochs $T$ do
    for each training example $\langle x, y\rangle$ do
        $agenda \leftarrow \{\langle\langle\rangle, x, 0\rangle\}$
        while $agenda$ contains unfinished items do
            $agenda' \leftarrow$ all possible successors of items in $agenda$
            Calculate the global score of every item in $agenda'$
            Drop all but the $b$ highest-scoring items and the highest-scoring finished item from $agenda'$
            if none of the items in $agenda'$ is correct then
                break                                                        ▷ early update
            else
                $agenda \leftarrow agenda'$
            end if
        end while
        $i* \leftarrow$ the item in $agenda$ with the highest global score
        $i' \leftarrow$ the *correct* item in $agenda$ with the highest global score
        for every possible feature-action combination $\langle f, a\rangle$ do
            $\omega(\langle f, a\rangle) \leftarrow \omega(\langle f, a\rangle) -$ the number of times $\langle f, a\rangle$ occurs in the history of $i*$
            $\omega(\langle f, a\rangle) \leftarrow \omega(\langle f, a\rangle) +$ the number of times $\langle f, a\rangle$ occurs in the history of $i'$      ▷ perceptron update
        end for
    end for
end for
```

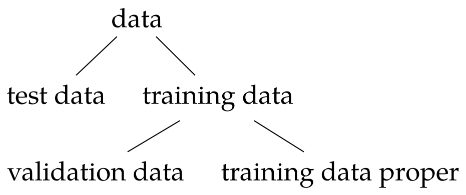


Figure 2: Splitting the data

The rest of the data is our training data in the broad sense. It is usually further subdivided into a larger set, the training data proper, and a smaller set, the *validation data* or *development data*. We can then train our system on the training data proper, evaluate it on the the validation data, make changes to it, train again and evaluate again to see if the changes were beneficial. Figure 2 shows an overview.

Especially if you have little data, it makes sense to perform the split between training data proper and validation data in multiple different ways (called *folds*), setting a different part of the training data aside as validation data each time. Training and validation is then performed separately for each fold, and the results are averaged.

References

Abzianidze, L., Bjerva, J., Evang, K., Haagsma, H., van Noord, R., Ludmann, P., Nguyen, D.-D., and Bos, J. (2017). The Parallel Meaning Bank: Towards a multilingual corpus of translations annotated with compositional meaning representations. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 242–247. Association for Computational Linguistics.

Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., and Schneider, N. (2013). Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186.

Bates, M., Boisen, S., and Makhoul, J. (1990). Developing an evaluation methodology for spoken language systems. In *Proceedings of the Third DARPA Speech and Natural Language Workshop*, pages 102–105.

Berant, J., Chou, A., Frostig, R., and Liang, P. (2013). Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544.

Bjerva, J., Bos, J., and Haagsma, H. (2016). The Meaning Factory at SemEval-2016 task 8: Producing AMRs with Boxer. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1179–1184.

Bos, J., Basile, V., Evang, K., Venhuizen, N., and Bjerva, J. (2017). The Groningen Meaning Bank. In Ide, N. and Pustejovsky, J., editors, *The Handbook of Linguistic Annotation*. Springer, Dordrecht.

Cai, Q. and Yates, A. (2013a). Large-scale semantic parsing via schema matching and lexicon extension. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 423–433. Association for Computational Linguistics.

Cai, Q. and Yates, A. (2013b). Semantic parsing Freebase: Towards open-domain semantic parsing. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 328–338.

Cai, S. and Knight, K. (2013). Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 748–752, Sofia, Bulgaria.

Clarke, J., Goldwasser, D., Chang, M.-W., and Roth, D. (2010). Driving semantic parsing from the world's response. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 18–27.

Evang, K. and Bos, J. (2016). Cross-lingual learning of an open-domain semantic parser. In *Proceedings of COLING 2016, the 25th International Conference on Computational Linguistics*.

Goldwasser, D., Reichart, R., Clarke, J., and Roth, D. (2011). Confidence driven unsupervised semantic parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 1486–1495.

Kamp, H. (1984). A Theory of Truth and Semantic Representation. In Groenendijk, J., Janssen, T. M., and Stokhof, M., editors, *Truth, Interpretation and Information*, pages 1–41. FORIS, Dordrecht, Holland/Cinnaminson, U.S.A.

Kamp, H. and Reyle, U. (1993). *From Discourse to Logic; An Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and DRT*. Kluwer, Dordrecht.

Kate, R. J., Wong, Y. W., and Mooney, R. J. (2005). Learning to transform natural to formal languages. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 1062–1068.

Krishnamurthy, J. and Mitchell, T. (2012). Weakly supervised training of semantic parsers. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 754–765.

Kwiatkowksi, T., Zettlemoyer, L., Goldwater, S., and Steedman, M. (2010). Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1223–1233.

Kwiatkowski, T., Choi, E., Artzi, Y., and Zettlemoyer, L. (2013). Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1545–1556. Association for Computational Linguistics.

Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., and Steedman, M. (2011). Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1512–1523.

Liang, P., Jordan, M., and Klein, D. (2011). Learning dependency-based compositional semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 590–599. Association for Computational Linguistics.

Papineni, K. A., Roukos, S., and Ward, T. R. (1997). Feature-based language understanding. In *Fifth European Conference on Speech Communication and Technology*.

Poon, H. and Domingos, P. (2009). Unsupervised semantic parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1–10.

Price, P. J. (1990). Evaluation of spoken language systems: the ATIS domain. In *Proceedings of the Third DARPA Speech and Natural Language Workshop*, pages 91–95.

Reddy, S., Lapata, M., and Steedman, M. (2014). Large-scale semantic parsing without question-answer pairs. *Transactions of the Association of Computational Linguistics*, 2:377–392.

Wang, Y., Berant, J., and Liang, P. (2015). Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342. Association for Computational Linguistics.

Zelle, J. M. and Mooney, R. J. (1996). Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the Eleventh National Conference of the American Association for Artificial Intelligence (AAAI-93)*, pages 817–822.

Zettlemoyer, L. and Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 658–666. AUAI Press.

Zettlemoyer, L. S. and Collins, M. (2007). Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687.