“What” is a very complex type: needs a noun and needs a S\NP to form a sentence. S\NP is basically a verb phrase (\textit{border Texas}).

These question are compositional: we can build bigger ones out of smaller pieces

- \textit{What states border Texas?}
- \textit{What states border states bordering Texas?}
- \textit{What states border states bordering states bordering Texas?}

In general, answering this does require parsing and not just slot-filling.
CCG Parsing

Show me flights to Prague

<table>
<thead>
<tr>
<th>S/N</th>
<th>λf.f</th>
<th>N</th>
<th>(N\N)/NP</th>
<th>NP PRG</th>
</tr>
</thead>
<tbody>
<tr>
<td>λx.f</td>
<td>λx.f(x) ∧ to(x, PRG)</td>
<td>λy.λf.λx.f(y) ∧ (x,y)</td>
<td>λx.λf.λx.f(x) ∧ (x, PRG)</td>
<td></td>
</tr>
</tbody>
</table>

‣ “to” needs an NP (destination) and N (parent)
‣ “Show me” is a no-op

Slide credit: Dan Klein

CCG Parsing

‣ Many ways to build these parsers
‣ One approach: run a “supertagger” (tags the sentence with complex labels), then run the parser

<table>
<thead>
<tr>
<th>What</th>
<th>states</th>
<th>border</th>
<th>Texas</th>
</tr>
</thead>
<tbody>
<tr>
<td>λx. state(x) ∧ borders(x, e89)</td>
<td>N</td>
<td>λx.λy.borders(y, x)</td>
<td>NP</td>
</tr>
<tr>
<td>(S/(S\NP))/N</td>
<td></td>
<td>(S\NP)/NP</td>
<td></td>
</tr>
<tr>
<td>λf.λg.λx.f(x) ∧ g(x)</td>
<td></td>
<td>λx.λy.borders(y, x)</td>
<td></td>
</tr>
</tbody>
</table>

‣ Parsing is easy once you have the tags, so we’ve reduced it to a (hard) tagging problem

Zettlemoyer and Collins (2005)

Training CCG Parsers

‣ Training data looks like pairs of sentences and logical forms

What states border Texas λx. state(x) ∧ borders(x, e89)
What borders Texas λx. borders(x, e89)
...

‣ What can we learn from these?
‣ Problem: we don’t know the derivation
  ‣ Texas corresponds to NP | e89 in the logical form (easy to figure out)
  ‣ What corresponds to (S/(S\NP))/N | λf.λg.λx. f(x) ∧ g(x)
  ‣ How do we infer that without being told it?

Lexicon

‣ GENLEX: takes sentence S and logical form L. Break up logical form into chunks C(L), assume any substring of S might map to any chunk

What states border Texas λx. state(x) ∧ borders(x, e89)

‣ Chunks inferred from the logic form based on rules:
  ‣ NP: e89 → (S\NP)/NP: λx. λy. borders(x, y)
  ‣ Any substring can parse to any of these in the lexicon
  ‣ Texas → NP: e89 is correct
  ‣ border Texas → NP: e89
  ‣ What states border Texas → NP: e89
  ...

Zettlemoyer and Collins (2005)
Learning

- Unsupervised learning of correspondences, like word alignment
- Iterative procedure: estimate “best” parses that derive each logical form, retrain the parser using these parses with supervised learning
- Eventually we converge on the right parses at the same time that we learn a model to build them

Jia and Liang (2016)

Seq2seq Semantic Parsing

Semantic Parsing as Translation

“what states border Texas”

$$\lambda x ( \text{state} (x) \text{and border} (x, e89))$$

- Write down a linearized form of the semantic parse, train seq2seq models to directly translate into this representation
- What are some benefits of this approach compared to grammar-based?
- What might be some concerns about this approach? How do we mitigate them?

Jia and Liang (2016)

Handling Invariances

“What states border Texas” “what states border Ohio”

- Parsing-based approaches handle these the same way
- Possible divergences: features, different weights in the lexicon
- Can we get seq2seq semantic parsers to handle these the same way?
- Key idea: don’t change the model, change the data
- “Data augmentation”: encode invariances by automatically generating new training examples
Data Augmentation

Jia and Liang (2016)

ROOT → (“what states border $STATE_ID$?”)
answer (NV, (state(V0), next_to (V0, NV), const (V0, stateid (STATE_ID))))
STATE_ID → (“texas”, “texas”)

- Lets us synthesize a “what states border ohio?” example
- Abstract out entities: now we can “remix” examples and encode invariance to entity ID. More complicated remixes too

Semantic Parsing as Translation

Geo

Jia and Liang (2016)

- Prolog
- Lambda calculus
- Other DSLs

Geo:
x: “what is the population of iowa?”
y: _answer ( NV , ( _population ( NV , V1 ) , _const ( V0 , _stateid ( iowa ) ) ) )

ATIS:
x: “can you list all flights from chicago to milwaukee”
y: ( _lambda $0 e ( _and ( _flight $0 ) ( _from $0 chicago : _ci ) ( _to $0 milwaukee : _ci ) ) )

Overnight:
x: “when is the weekly standup”
y: ( call listValue ( call getProperty meeting.weekly_standup ( string start_time ) ) )

- Handle all of these with uniform machinery!

Semantic Parsing as Translation

Jia and Liang (2016)

Applications

- GeoQuery (Zelle and Mooney, 1996): answering questions about states (~80% accuracy)
- Jobs: answering questions about job postings (~80% accuracy)
- ATIS: flight search
- Can do well on all of these tasks if you handcraft systems and use plenty of training data: these domains aren’t that rich

<table>
<thead>
<tr>
<th>Previous Work</th>
<th>Geo</th>
<th>ATIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zettlemoyer and Collins (2007)</td>
<td>85.0</td>
<td>76.3</td>
</tr>
<tr>
<td>Kwiatkowski et al. (2010)</td>
<td>88.9</td>
<td>84.6</td>
</tr>
<tr>
<td>Liang et al. (2011)</td>
<td>91.1</td>
<td>94.3</td>
</tr>
<tr>
<td>Kwiatkowski et al. (2011)</td>
<td>88.6</td>
<td>82.8</td>
</tr>
<tr>
<td>Poon (2013)</td>
<td>83.5</td>
<td>83.5</td>
</tr>
<tr>
<td>Zhao and Huang (2015)</td>
<td>88.9</td>
<td>84.2</td>
</tr>
</tbody>
</table>

- Three forms of data augmentation all help
- Results on these tasks are still not as strong as hand-tuned systems from 10 years ago, but the same simple model can do well at all problems
**Regex Prediction**

- Can use for other semantic parsing-like tasks
- Predict regex from text

Problem: requires a lot of data: 10,000 examples needed to get ~60% accuracy on pretty simple regexes

Locascio et al. (2016)

**SQL Generation**

- Convert natural language description into a SQL query against some DB
- How to ensure that well-formed SQL is generated?
  - Three seq2seq models
- How to capture column names + constants?
  - Pointer mechanisms

Zhong et al. (2017)