Scaling Semantic Parsers with On-the-Fly Ontology Matching

Semantic Parsing

Q: How many people live in Seattle?

MR: $\lambda x.\text{population}(\text{seattle}, x)$

620,778
Q: Who managed Liverpool F.C. from 2004 to June 2010?
A: Rafael Benitez

Q: What architectural style is the Brooklyn Bridge?
A: Gothic Revival architecture

Q: What are the symptoms of prostate cancer?
A: {Hematuria, Nocturia, Dysuria, ... }

Q: How many people ride the monorail in Seattle daily?
A: 7,000
Open Domain QA

Freebase is a community authored knowledge base with:

- 40 Million Entities
- 2 Billion Facts
- 20,000 Relations
- 10,000 Types
- 100 Domains

Q: Who managed Liverpool FC from 2004 to June 2010?
A: Rafael Benitez
How many people live in Seattle?
Query is Domain Dependent

How many people live in Seattle?

\[ \lambda x. \text{eq}(x, \text{count}(\lambda y. \text{person}(y) \land \text{home}(y, \text{seattle}))) \]

<table>
<thead>
<tr>
<th>Person</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eunsol</td>
<td>Seattle</td>
</tr>
<tr>
<td>Luke</td>
<td>Seattle</td>
</tr>
<tr>
<td>Jane</td>
<td>Boston</td>
</tr>
</tbody>
</table>
Query is Domain Dependent

How many people live in Seattle?

\[ \lambda x. eq(x, count(\lambda y. person(y) \land home(y, seattle))) \]

\[ \lambda x. population(seattle, x) \]

<table>
<thead>
<tr>
<th>City</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>620778</td>
</tr>
<tr>
<td>Boston</td>
<td>636479</td>
</tr>
</tbody>
</table>
How many people live in Seattle?

<table>
<thead>
<tr>
<th>City</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>620778</td>
</tr>
<tr>
<td>Boston</td>
<td>636479</td>
</tr>
</tbody>
</table>

- Requires different syntax for different domains
  - Grammars do not generalize well
  - Grammars are hard to learn
Query is Domain Dependent

How many people live in Seattle?

\[ \lambda x. \text{population}(\text{seattle}, x) \]

<table>
<thead>
<tr>
<th>City</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>450,000</td>
</tr>
<tr>
<td>Boston</td>
<td>750,000</td>
</tr>
</tbody>
</table>

How many people have won the Nobel peace prize?

\[ \lambda x. \text{eq}(x, \text{count}(\lambda y. \text{person}(y) \land \text{won}(y, \text{nobel_peace_prize}))) \]

<table>
<thead>
<tr>
<th>Person</th>
<th>Award</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nelson M.</td>
<td>Nobel P.P.</td>
</tr>
<tr>
<td>Mother T.</td>
<td>Nobel P.P.</td>
</tr>
<tr>
<td>Leymah G.</td>
<td>Nobel P.P.</td>
</tr>
</tbody>
</table>
Query is Domain Dependent

How many people live in Seattle?
\[ \lambda x. \text{population}(\text{seattle}, x) \]

How many people have won the Nobel peace prize?
\[ \lambda x. \text{eq}(x, \text{count}(\lambda y. \text{person}(y) \land \text{won}(y, \text{Nobel peace prize})))) \]
How many people live in Seattle?
2 Stage Semantic Parsing

1. Domain independent, linguistically motivated parse.

How many people live in Seattle?

\[ \lambda x. eq(x, count(\lambda y. people(y) \land \exists ev. live(y, ev) \land in(seattle, ev))) \]
2 Stage Semantic Parsing

2. Domain specific ontology match.

How many people live in Seattle?

\[ \lambda x. eq(x, count(\lambda y. person(y) \land home(y, seattle))) \]
How many people live in Seattle?

\[ \lambda x. \text{eq}(x, \text{count}(\lambda y. \text{person}(y) \land \text{home}(y, \text{seattle}))) \]

\[ \Rightarrow \lambda x. \text{population}(\text{seattle}, x) \]
2 Stage Semantic Parsing

2. Domain specific ontology match.

How many people live in Seattle?

- All domains use same syntax that generalizes well
- Ontology match can be guided by the structure of the underspecified logical form
Overview

• 2 stage semantic parsing
  ➡ Domain independent parsing
  ➡ Domain dependent ontology matching
• Modeling and Inference
• Learning from Question/Answer pairs
• Experiments
2 Stage Semantic Parsing

1. Domain independent, linguistically motivated parse.
2. Domain specific ontology match.

How many people live in Seattle?

\[ \lambda x. eq(x, \text{count}(\lambda y. \text{person}(y) \land \exists v. \text{live}(y, v) \land \text{in}(\text{seattle}, v))) \]

\[ \lambda x. eq(x, \text{count}(\lambda y. \text{person}(y) \land \text{home}(y, \text{seattle}))) \]

\[ \lambda x. \text{population}(	ext{seattle}, x) \]

<table>
<thead>
<tr>
<th>City</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>620778</td>
</tr>
<tr>
<td>Boston</td>
<td>636479</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eunsol</td>
<td>Seattle</td>
</tr>
<tr>
<td>Luke</td>
<td>Seattle</td>
</tr>
<tr>
<td>Jane</td>
<td>Boston</td>
</tr>
</tbody>
</table>
2 Stage Semantic Parsing

Domain Independent Parse

Domain Independent Parse

Ontology Match

Ontology Match

Ontology Match

Structure Match

x.how many people live in seattle, x
## 2 Stage Semantic Parsing

### Domain Independent Parse

<table>
<thead>
<tr>
<th>How many people live in Seattle</th>
<th>Seattle</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S/(S/NP)/N )</td>
<td></td>
</tr>
<tr>
<td>( \lambda f \lambda g \lambda x. eq(x, count(\lambda y.g(y) \land f(y))) )</td>
<td></td>
</tr>
<tr>
<td>( N )</td>
<td></td>
</tr>
<tr>
<td>( \lambda x. people(x) )</td>
<td></td>
</tr>
<tr>
<td>( S/)NP</td>
<td></td>
</tr>
<tr>
<td>( \lambda x \lambda ev. live(x, ev) )</td>
<td></td>
</tr>
<tr>
<td>( S/)NP</td>
<td></td>
</tr>
<tr>
<td>( \lambda x \lambda f \exists ev. in(ev, x) \land f(ev) )</td>
<td></td>
</tr>
<tr>
<td>( S/)NP</td>
<td></td>
</tr>
<tr>
<td>( \lambda x eq(x, count(\lambda y. \exists ev. people(y) \land live(y, ev) \land in(ev, seattle))) )</td>
<td></td>
</tr>
</tbody>
</table>

### Ontology Match
2 Stage Semantic Parsing

Domain Independent Parse

<table>
<thead>
<tr>
<th>How many</th>
<th>people</th>
<th>live</th>
<th>in</th>
<th>Seattle</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S/(S\backslash NP)/N$</td>
<td>$N$</td>
<td>$S\backslash NP$</td>
<td>$S\backslash S/\backslash NP$</td>
<td>$NP$</td>
</tr>
<tr>
<td>$\lambda f \lambda g \lambda x. eq(x, \text{count}(\lambda y. g(y) \land f(y)))$</td>
<td>$\lambda x. people(x)$</td>
<td>$\lambda x \lambda x. ev.\text{live}(x, ev)$</td>
<td>$\lambda x \lambda f \exists ev. \text{in}(ev, x) \land f(ev)$</td>
<td>$\text{seattle}$</td>
</tr>
<tr>
<td>$\lambda x. eq(x, \text{count}(\lambda y. \exists ev. people(y) \land \text{live}(y, ev) \land \text{in}(ev, \text{seattle})))$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ontology Match

$\lambda x. eq(x, \text{count}(\lambda y. \exists ev. people(y) \land \text{live}(y, ev) \land \text{in}(ev, \text{seattle})))$

Structure Match

$\lambda x. \text{how}\_\text{many}\_\text{people}\_\text{live}\_\text{in}(\text{seattle}, x)$

Constant Matches for $\equiv$

$\lambda x. \text{how}\_\text{many}\_\text{people}\_\text{live}\_\text{in}(\text{seattle}, x)$

$\lambda x. \text{population}(\text{seattle}, x)$
CCG Parsing

How many \( \vdash S/(S\backslash NP)/N \) : \( \lambda f \lambda g \lambda x. eq(x, \text{count}(\lambda y. f(y) \land g(y))) \)

people \( \vdash N \) : \( \lambda x. people(x) \)

live \( \vdash S\backslash NP \) : \( \lambda x \lambda e v. live(x, e v) \)

in \( \vdash S\backslash S/NP \) : \( \lambda x \lambda f \exists e v. f(e v) \land \text{in}(e v, x) \)

Seattle \( \vdash NP \) : \( \text{seattle} \)

\[ \lambda g \lambda x. eq(x, \text{count}(\lambda y. people(y) \land g(y))) \]

How many \( \vdash S/(S\backslash NP)/N \) : \( \lambda f \lambda g \lambda x. eq(x, \text{count}(\lambda y. f(y) \land g(y))) \)

people \( \vdash N \) : \( \lambda x. people(x) \)

live \( \vdash S\backslash NP \) : \( \lambda x \lambda e v. live(x, e v) \)

in \( \vdash S\backslash S/NP \) : \( \lambda x \lambda f \exists e v. f(e v) \land \text{in}(e v, x) \)

Seattle \( \vdash NP \) : \( \text{seattle} \)
CCG Parsing

Do not have lexicon for every domain

- Use Wiktionary for syntactic cues
- Parse to *underspecified* semantics

Lexicon

\[
\begin{align*}
\text{in} & \vdash S/S/NP : \lambda x \lambda f \exists ev.f(ev) \wedge \text{in}(ev,x) \\
\text{Seattle} & \vdash NP : seattle
\end{align*}
\]
### Domain Independent Parsing

<table>
<thead>
<tr>
<th>Word</th>
<th>Syntax</th>
<th>Underspecified semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>How many</td>
<td>$S/(S\backslash NP)/N$</td>
<td>$\lambda f \lambda g \lambda x. eq(x, count(\lambda y.f(y) \land g(y)))$</td>
</tr>
<tr>
<td>What</td>
<td>$S/(S\backslash NP)/N$</td>
<td>$\lambda f \lambda g \lambda x. f(x) \land g(x)$</td>
</tr>
<tr>
<td>most</td>
<td>$NP/N$</td>
<td>$\lambda f.\text{max_count}(\lambda y.f(y))$</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How many people live in Seattle

\[
\frac{S/(S\setminus NP)/N}{\lambda f \lambda g \lambda x. eq(x, count(\lambda y. g(y) \land f(y))))}
\]
## Domain Independent Parsing

How many people live in Seattle

<table>
<thead>
<tr>
<th>How many</th>
<th>people</th>
<th>live</th>
<th>in</th>
<th>Seattle</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S/(S\backslash NP)/N$</td>
<td>$\lambda f \lambda g \lambda x. eq(x, count(\lambda y.g(y) \land f(y)))$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Domain Independent Parsing

#### 49 domain independent lexical items:

<table>
<thead>
<tr>
<th>Word</th>
<th>Syntax</th>
<th>Underspecified semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>How many</td>
<td>( S/(S\setminus NP)/N )</td>
<td>( \lambda f \lambda g \lambda x . eq(x, count(\lambda y . f(y) \land g(y))) )</td>
</tr>
<tr>
<td>What</td>
<td>( S/(S\setminus NP)/N )</td>
<td>( \lambda f \lambda g \lambda x . f(x) \land g(x) )</td>
</tr>
<tr>
<td>most</td>
<td>( NP/N )</td>
<td>( \lambda f . max_count(\lambda y . f(y)) )</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 56 underspecified lexical categories:

<table>
<thead>
<tr>
<th>Part-of-Speech</th>
<th>Syntax</th>
<th>Underspecified semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>proper noun</td>
<td>( NP )</td>
<td>( C )</td>
</tr>
<tr>
<td>noun</td>
<td>( N )</td>
<td>( \lambda x . P(x) )</td>
</tr>
<tr>
<td>noun</td>
<td>( N/N )</td>
<td>( \lambda f \lambda x . f(x) \land P(x) )</td>
</tr>
<tr>
<td>verb</td>
<td>( S\setminus NP )</td>
<td>( \lambda \lambda ev . P(x, ev) )</td>
</tr>
<tr>
<td>verb</td>
<td>( S\setminus NP/NP )</td>
<td>( \lambda x \lambda y \lambda \lambda ev . P(y, x, ev) )</td>
</tr>
<tr>
<td>preposition</td>
<td>( N\setminus N/NP )</td>
<td>( \lambda f \lambda x \lambda y . P(y, x) \land f(y) )</td>
</tr>
<tr>
<td>preposition</td>
<td>( S\setminus S/NP )</td>
<td>( \lambda f \lambda x \lambda \exists ev . P(ev, x) \land f(ev) )</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Domain Independent Parsing

<table>
<thead>
<tr>
<th>How many</th>
<th>people</th>
<th>live</th>
<th>in</th>
<th>Seattle</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S/(S\backslash NP)/N$</td>
<td>$N$</td>
<td>$S\backslash NP$</td>
<td>$S\backslash S/NP$</td>
<td>$NP$</td>
</tr>
<tr>
<td>$\lambda f \lambda g \lambda x. eq(x, count(\lambda y.g(y) \land f(y)))$</td>
<td>$\lambda x.P(x)$</td>
<td>$\lambda x \lambda ev.P(x, ev)$</td>
<td>$\lambda x \lambda f \exists ev.P(ev, x) \land f(ev)$</td>
<td>$C$</td>
</tr>
</tbody>
</table>
How many people live in Seattle

\[
\begin{array}{c}
S/(S\backslash NP)/N \\
\lambda f \lambda g \lambda x. eq(x, count( \lambda y. g(y) \land f(y))))
\end{array}
\]

How many people live in Seattle

\[
\begin{array}{c}
S/(S\backslash NP) \\
\lambda g \lambda x. eq(x, count(\lambda y. g(y) \land P(y))))
\end{array}
\]
Domain Independent Parsing

How many people live in Seattle

\[ S/(S \setminus NP)/N \] \( \lambda f \lambda g \lambda x. \text{eq}(x, \text{count}(\lambda y. g(y) \land f(y))) \)

\[ N \] \( \lambda x. P(x) \)

\[ S/(S \setminus NP) \] \( \lambda g \lambda x. \text{eq}(x, \text{count}(\lambda y. g(y) \land P(y))) \)

\[ S/(S \setminus NP) \] \( \lambda x. \text{eq}(x, \text{count}(\lambda y. P(y) \land P(y))) \)

\[ S \setminus NP \] \( \lambda x \lambda \exists \text{ev} . P(x, \text{ev}) \land f(\text{ev}) \)

\[ S \setminus S \] \( \lambda f \exists \text{ev} . P(\text{ev}, C) \land f(\text{ev}) \)

\[ S \setminus S \] \( \lambda x \exists \text{ev} . P(x, \text{ev}) \land P(\text{ev}, C) \)

\( \lambda x. \text{eq}(x, \text{count}(\lambda y. P(y) \land \exists \text{ev} . P(y, \text{ev}) \land P(\text{ev}, C))))) \)

\( \lambda x. \text{eq}(x, \text{count}(\lambda y. g(y) \land f(y))) \)
Domain Independent Parsing

String labels signify source words, not semantic constants.
2 Step Semantic Parsing

Domain Independent Parse

\[
\lambda x. eq(x, \text{count}(\lambda y. g(y) \land f(y))) \rightarrow \lambda x. \text{people}(x) \rightarrow S/NP \rightarrow S/NP \rightarrow S/S/NP \rightarrow \text{in} \rightarrow \text{Seattle} \rightarrow N/P
\]

\[
\lambda x. eq(x, \text{count}(\lambda y. \exists ev. \text{people}(y) \land \text{live}(y, ev) \land \text{in}(ev, \text{seattle})))
\]

Ontology Match

Structure Match

\[
\lambda x. \text{how} \_ \text{many} \_ \text{people} \_ \text{live} \_ \text{in}(\text{seattle}, x)
\]

Constant Matches for \( \equiv \)

\[
\lambda x. \text{population}(\text{seattle}, x)
\]
Structural Match

Collapse and expand subexpressions in underspecified logical form with operators that:

1. Collapse simple typed sub-expression
2. Collapse complex typed sub-expression
3. Expand predicate

New example

How many people ride the monorail in Seattle daily?

\[ \lambda x. eq(x, \text{count}(\lambda y. \text{people}(y) \land \exists e. \text{ride}(y, \text{\_z. monorail}(z) \land \text{in}(z, \text{seattle}), e) \land \text{daily}(e))) \]

\[ \equiv \lambda x. \text{transit system/daily riders(seattle monorail, x)} \]
Structural Match

1. Find subexpression with type allowed in KB

\[ \lambda x. eq(x, \text{count}(\lambda y. \text{people}(y) \land \exists e. \text{ride}(y, iz. \text{monorail}(z) \land \text{in}(z, \text{seattle}), e) \land \text{daily}(e))) \]

2. Replace with new underspecified constant
1. Find subexpression with type allowed in KB

\[ \lambda x. eq(x, \text{count}(\lambda y. \text{people}(y) \land \exists e. \text{ride}(y, \text{monorail}(z) \land \text{in}(z, \text{seattle}), e) \land \text{daily}(e))) \]

2. Replace with new underspecified constant

\[ \lambda x. eq(x, \text{count}(\lambda y. \text{people}(y) \land \exists e. \text{ride}(y, \text{monorail in seattle}, e) \land \text{daily}(e))) \]
Structural Match

1. Find subexpression with type allowed in KB

\[ \lambda x. eq(x, count(\lambda y. people(y) \land \exists e. ride(y, \exists z. monorail(z) \land in(z, seattle), e) \land daily(e))) \]

integer typed subexpression

2. Replace with new underspecified constant

\[ \lambda x. eq(x, \text{how many people ride daily the monorail in seattle}) \]
Structural Match

1. Find subexpression with type allowed in KB

\[ \lambda x. eq(x, \text{count}(\lambda y. \text{people}(y) \land \exists e. \text{ride}(y, \uparrow z. \text{monorail}(z) \land \text{in}(z, \text{seattle}), e) \land \text{daily}(e))) \]

integer typed subexpression

2. Replace with new underspecified constant

\[ \lambda x. eq(x, \text{count}(\lambda y. \text{people}(y) \land \exists e. \text{ride}(y, \text{monorail\_in\_seattle}, e) \land \text{daily}(e))) \]

\[ \lambda x. eq(x, \text{how\_many\_people\_ride\_daily\_the\_monorail\_in\_seattle}) \]
Structural Match

1. Find subexpression with type allowed in KB

\[ \lambda x. eq(x, \text{count}(\lambda y. \text{people}(y) \land \exists e. \text{ride}(y, \text{monorail}(z) \land \text{in}(z, \text{seattle}), e) \land \text{daily}(e))) \]

2. Replace with new underspecified constant

\[ \lambda x. eq(x, \text{how\_many\_people\_ride\_daily\_the\_monorail\_in\_seattle}) \]
Structural Match

1. Find subexpression with type allowed in KB

2. Replace with new underspecified constant

\[ \lambda x. eq(x, count(\lambda y. people(y) \land \exists e. ride(y, monorail\_in\_seattle), e) \land daily(e))) \]
Structural Match

1. Find subexpression with type allowed in KB

\[ \lambda x. eq(x, count(\lambda y. people(y) \land \exists e. ride(y, monorail\_in\_seattle), e) \land daily(e))) \]

2. Replace with new underspecified constant
Structural Match

1. Find subexpression with type allowed in KB

$$\lambda x. eq(x, count(\lambda y. people(y) \land \exists e. ride(y, monorail\_in\_seattle), e) \land daily(e)))$$

2. Replace with new underspecified constant

$$\lambda x. eq(x, count(\lambda y. people(y) \land \text{ride\_daily}(y, monorail\_in\_seattle)))$$
Structural Match

1. Find subexpression with type allowed in KB

\[ \lambda x. eq(x, \text{count}(\lambda y. \text{people}(y) \land \exists e. \text{ride}(y, \text{monorail\_in\_seattle}, e) \land \text{daily}(e))) \]

2. Replace with new underspecified constant

\[ \lambda x. eq(x, \text{count}(\lambda y. \text{people}(y) \land \text{ride\_daily}(y, \text{monorail\_in\_seattle}), e) \land \text{daily}(e))) \]

\[ \lambda x. \text{how\_many\_people\_ride\_daily}(\text{the\_monorail\_in\_seattle}, x) \]
Structural Match

1. Find subexpression with type allowed in KB

$$\lambda x. eq(x, count(\lambda y. people(y) \land \exists e. ride(y, monorail_in_seattle), e) \land daily(e)))$$

2. Replace with new underspecified constant

$$\lambda x. eq(x, count(\lambda y. people(y) \land ride_daily(y, monorail_in_seattle)))$$

$$\lambda x. how_many_people_ride_daily(the_monorail_in_seattle, x)$$
Structural Match

1. Find subexpression with type allowed in KB

\[ \lambda x. \text{eq}(x, \text{count}(\lambda y. \text{people}(y) \land \exists e. \text{ride}(y, \text{monorail\_in\_seattle}, e) \land \text{daily}(e))) \]

\[ \lambda x. \text{how\_many\_people\_ride\_daily}(\text{the\_monorail\_in\_seattle}, x) \]
Structural Match

\[ \lambda x. \text{how\_many\_people\_ride\_daily} (\text{the\_monorail\_in\_seattle}, x) \]
Constant Match

Replace constants with constants from KB

\[ \lambda x. \text{how\_many\_people\_ride\_daily}(\text{the\_monorail\_in\_seattle}, x) \]

Assume constants have English string labels!
Constant Match

Replace constants with constants from KB

\[ \lambda x. \text{how\_many\_people\_ride\_daily}(\text{the\_monorail\_in\_seattle}, x) \]

\[ \approx \lambda x. \text{transit\_system\_daily\_riders}(\text{seattle\_monorail}, x) \]
Constant Match

Replace constants with constants from KB

\[ \lambda x. \text{how\_many\_people\_ride\_daily}(\text{the\_monorail\_in\_seattle}. x) \]

\[ \approx \lambda x. \text{how\_many\_people\_ride\_daily}(\text{seattle\_monorail}, x) \]

\[ \approx \lambda x. \text{transit\_system/daily\_riders}(\text{seattle\_monorail}, x) \]
Constant Match

Replace constants with constants from KB

\[ \lambda x.\text{how\_many\_people\_ride\_daily}(\text{the\_monorail\_in\_seattle.}\_x) \]

\[ \Rightarrow \]

\[ \lambda x.\text{how\_many\_people\_ride\_daily}(\text{seattle\_monorail},x) \]

\[ \Rightarrow \]

\[ \lambda x.\text{transit\_system/daily\_riders}(\text{seattle\_monorail},x) \]

\[ \Rightarrow \lambda x.\text{transit\_system/daily\_riders}(\text{seattle\_monorail},x) \]
Overview

- 2 stage semantic parsing
  - Domain independent parsing
  - Domain dependent ontology matching

- Modeling and Inference
- Learning from Question/Answer pairs
- Experiments
2 Stage Semantic Parsing

Domain Independent Parse

Ontology Match

Structure Match

Constant Matches for $\equiv$
Scoring Derivations

Derivations $d$ are scored using a linear model:

$$score(d) = \phi(d)\theta$$

with feature vector $\phi(d)$ that decomposes over:

- Domain independent parse
- Structural match
- Constant match
  - Lexical features
  - Knowledge base features
Lexical Features

When did Prairie Home Companion first air?
\[ \lambda x.\text{when\_first\_air}(\text{prairie\_home\_companion}, x) \]
\[ \approx \lambda x.\text{radio\_program\_first\_broadcast}(\text{prairie\_home\_companion}, x) \]

How high is Niagara Falls?
\[ \lambda x.\text{high}(\text{niagara\_falls}, x) \]
\[ \approx \lambda x.\text{location\_geocode\_elevation}(\text{niagara\_falls}, x) \]

- Exact string match
- Stemmed string match
- Synonym match
- Wiktionary gloss overlap
Lexical Features

elevation (plural elevations)
1. The act of raising from a lower place, condition, or quality to a higher; said of material things, persons, the mind, the voice, etc.; as, the elevation of grain; elevation to a throne; elevation to sainthood; elevation of mind, thoughts, or character.
2. The condition of being or feeling elevated; heightened; exaltation.

high (comparative higher, superlative highest)
1. Being elevated in position or status, a state of being above many things. [quotations ▼]

\[\lambda x.\text{high}(\text{niagara_falls}, x)\]
Test logical structure to see if it can exist in knowledge base.
Test logical structure to see if it can exist in knowledge base.
Knowledge Base Features

Test logical structure to see if it can exist in knowledge base.
Inference

Ontology matching step is:

- Exponential in arity of most complex predicate
- Polynomial in number of logical symbols
- Linear in the size of the knowledge base

- Use dynamic programming
- Prune heavily according to local score
- Call constant matching operators greedily
2 Stage Semantic Parsing

Domain Independent Parse

Ontology Match

\[ \lambda x.\text{eq}(x, \text{count}(\lambda y.\text{g}(y) \land f(y))) \]

\[ \lambda x.\text{people}(x) \rightarrow \lambda x.\text{live}(x, ev) \rightarrow \lambda x.\text{in}(ev, \text{seattle}) \]

\[ \lambda x.\text{eq}(x, \text{count}(\lambda y.\exists ev.\text{people}(y) \land \text{live}(y, ev) \land \text{in}(ev, \text{seattle}))) \]

\[ \lambda x.\text{how\_many\_people\_live\_in}(\text{seattle}, x) \]

\[ \lambda x.\text{population}(\text{seattle}, x) \]
Overview

- 2 stage semantic parsing
  - Domain independent parsing
  - Domain dependent ontology matching
- Modeling and Inference
- Learning from Question/Answer pairs
- Experiments
Learning

Input

Q/A pairs \( \{(x_i, a_i) : i = 1, \ldots, n\} \)
Knowledge Base, Wiktionary, Underspecified Lexicon

Algorithm

For \( i = 1, \ldots, n \):

\[
C \leftarrow \text{Max scoring correct parses of } x_i
\]

\[
W \leftarrow \text{Margin violating incorrect parses of } x_i
\]

\[
\theta = \theta + \frac{1}{|C|} \sum_{c \in C} \phi(c) - \frac{1}{|W|} \sum_{w \in W} \phi(w)
\]
Overview

• 2 stage semantic parsing
  ➞ Domain independent parsing
  ➞ Domain dependent ontology matching
• Modeling and Inference
• Learning from Question/Answer pairs
• Experiments
Experiments

Q/A from databases - evaluate on exact answer match

**Freebase917**
- 642 training sentences, 275 test sentences
- 135 million facts, 18 million entities, 2000 relations

**GeoQuery**
- 600 training sentences, 280 test sentence
- 14018 facts, 3839 entities, 12 relations
- High degree of compositional complexity

Feature initialization:
- **Lexical** - Prefer exact and partial string match
- **Knowledge base** - Prefer concepts in knowledge base
Related Work

Learning From Q/A Pairs:
- Clarke et.al. 2010
- Goldwasser et.al. 2011
- Liang et.al. 2011 (DCS)
- Berant et.al. 2013

Learning CCG from Labelled Logical Forms:
- Kwiatkowski et.al. 2011 (FUBL)
- Cai & Yates 2013
## Related Work

### Learning From Q/A Pairs:
- Clarke et. al. 2010
- Goldwasser et. al. 2011
- Liang et. al. 2011 (DCS)
- Berant et. al. 2013

### Learning CCG from Labelled Logical Forms:
- Kwiatkowski et. al. 2011 (FUBL)
- Cai & Yates 2013
Experiments

Freebase 917

Cai & Yates  
Berant et.al.  
Our Approach

50  
54  
58  
62  
66  
70
Experiments

GeoQuery

<table>
<thead>
<tr>
<th>Method</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUBL</td>
<td>90</td>
</tr>
<tr>
<td>DCS</td>
<td>85</td>
</tr>
<tr>
<td>DCS with L+</td>
<td>95</td>
</tr>
<tr>
<td>Our Approach</td>
<td>90</td>
</tr>
</tbody>
</table>
Example Parses

How many operating systems is Adobe Flash compatible with?

\[ \lambda x. \text{eq}(x, \text{count}(\lambda y. \text{software_compatibility.operating_system(adobe_flash, y)))) \]
Example Parses

Who is the CEO of Save-A-Lot?

\[ \lambda x. \text{person}(x) \land \exists y. \text{organization}(y, \text{savealot}) \land \text{board_member.leader_of}(x, y) \land \text{leadership.role}(y, \text{ceo}) \]
Example Errors

How many children does Jerry Seinfeld have?

Target:

\[ \lambda x. \text{eq}(x, \text{count}(\lambda y. \text{person}.\text{children}(\text{jerry}\_\text{seinfeld}, y))) \]

Prediction:

\[ \lambda x. \text{eq}(x, \text{count}(\lambda y. \text{person}.\text{children}(y, \text{jerry}\_\text{seinfeld}))) \]
What programming languages were used for AOL instant messenger?

Target:
\[ \lambda x.\text{languages\_used(aol\_instant\_messenger, x)} \]

Prediction:
\[ \lambda x.\text{languages\_used(aol\_instant\_messenger, x)} \]
\[ \wedge \text{programming\_language(x)} \]
Future Work

Information Extraction:

Alan Turing was a British mathematician, logician, cryptanalyst, and computer scientist.

\[\text{nationality}(\text{AT, UK}) \land \text{notable\_for}(\text{AT, mathematian}) \land \text{profession}(\text{AT, logic}) \land \text{research}(\text{AT, cryptanalysm}) \land \text{notable\_type}(\text{AT, compsci})\]

Multiple databases:

Who’s going to play Batman in the next movie?
When’s the next flight from Seattle to London?
What’s the best restaurant near the Grand Hyatt?

Parser

KAYAK  yelp  IMDb  DBpedia
Questions

?