

# Scaling Semantic Parsers with On-the-Fly Ontology Matching

Tom Kwiakowski, Eunsol Choi, Yoav Artzi, Luke Zettlemoyer.



# Semantic Parsing

Q: How many people live in Seattle?



Semantic Parser



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



620,778

# Open Domain QA

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

Q Who managed Liverpool FC from 2004 to June 2010?

A  **Freebase** is a community authored knowledge base with:

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

Q

A

7,000

/?

# Query is Domain Dependent

How many people live in Seattle?

# Query is Domain Dependent

How many people live in Seattle?




$\lambda x. \text{eq}(x, \text{count}(\lambda y. \text{person}(y) \wedge \text{home}(y, \text{seattle})))$


Person	Home
Eunsol	Seattle
Luke	Seattle
Jane	Boston

A stack of four white, circular disks or coins, positioned to the right of the table.


# Query is Domain Dependent

How many people live in Seattle?


  $\lambda x. \text{eq}(x, \text{count}(\lambda y. \text{person}(y) \wedge \text{home}(y, \text{seattle})))$

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

Person	Home
Eunsol	Seattle
Luke	Seattle
Jane	Boston



City	Population
Seattle	620778
Boston	636479



# Query is Domain Dependent

How many people live in Seattle?

- Requires different syntax for different domains
  - Grammars do not generalize well
  - Grammars are hard to learn

e)))




$\lambda x. \text{population}(\text{Seattle}, x)$

Jane

Boston




City	Population
Seattle	620778
Boston	636479






# Query is Domain Dependent


How many people live in Seattle?

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


City	Population
Seattle	450,000
Boston	750,000



How many people have won the Nobel peace prize?

  $\lambda x. \text{eq}(x, \text{count}(\lambda y. \text{person}(y) \wedge \text{won}(y, \text{nobel\_peace\_prize})))$

Person	Award
Nelson M.	Nobel P.P.
Mother T.	Nobel P.P.
Leymah G.	Nobel P.P.




# Query is Domain Dependent


How many people live in Seattle?

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


City	Population
Seattle	620778
Boston	636479



How many people have won the Nobel peace prize?

  $\lambda x. \text{eq}(x, \text{count}(\lambda y. \text{person}(y) \wedge \text{won}(y, \text{nobel\_peace\_prize})))$

Person	Award
Nelson M.	Nobel P.P.
Mother T.	Nobel P.P.
Leymah G.	Nobel P.P.



# 2 Stage Semantic Parsing

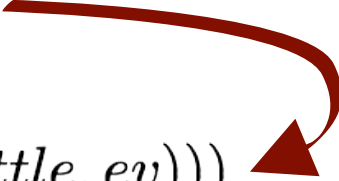
How many people live in Seattle?

# 2 Stage Semantic Parsing

I. Domain independent, linguistically motivated parse.

How many people live in Seattle?

$\lambda x.eq(x, count(\lambda y.people(y) \wedge \exists ev.live(y, ev) \wedge in(seattle, ev)))$



# 2 Stage Semantic Parsing

## 2. Domain specific ontology match.


How many people live in Seattle?

$\lambda x.eq(x, count(\lambda y.people(y) \wedge \exists ev.live(y, ev) \wedge in(seattle, ev)))$



$\lambda x.eq(x, count(\lambda y.person(y) \wedge home(y, seattle)))$

Person	Home
Eunsol	Seattle
Luke	Seattle
Jane	Boston

A stack of four grey, cylindrical disks, representing a database or data storage.

# 2 Stage Semantic Parsing

## 2. Domain specific ontology match.

How many people live in Seattle?

$\lambda x.eq(x, count(\lambda y.people(y) \wedge \exists ev.live(y, ev) \wedge in(seattle, ev)))$



$\lambda x.eq(x, count(\lambda y.person(y) \wedge home(y, seattle)))$



$\lambda x.population(seattle, x)$

City	Population
Seattle	620778
Boston	636479

Person	Home
Eunsol	Seattle
Luke	Seattle
Jane	Boston

# 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

$\lambda x. \text{population}(\text{Seattle}, x)$

City	Population
Seattle	620778
Boston	636479



Luke	Seattle
Jane	Boston



# 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?

$\lambda x. eq(x, count(\lambda y. people(y) \wedge \exists ev. live(y, ev) \wedge in(seattle, ev)))$




$\lambda x. eq(x, count(\lambda y. person(y) \wedge home(y, seattle)))$




$\lambda x. population(seattle, x)$

City	Population
Seattle	620778
Boston	636479



Person	Home
Eunsol	Seattle
Luke	Seattle
Jane	Boston



# 2 Stage Semantic Parsing

Domain Independent Parse

Domain Independent Parse

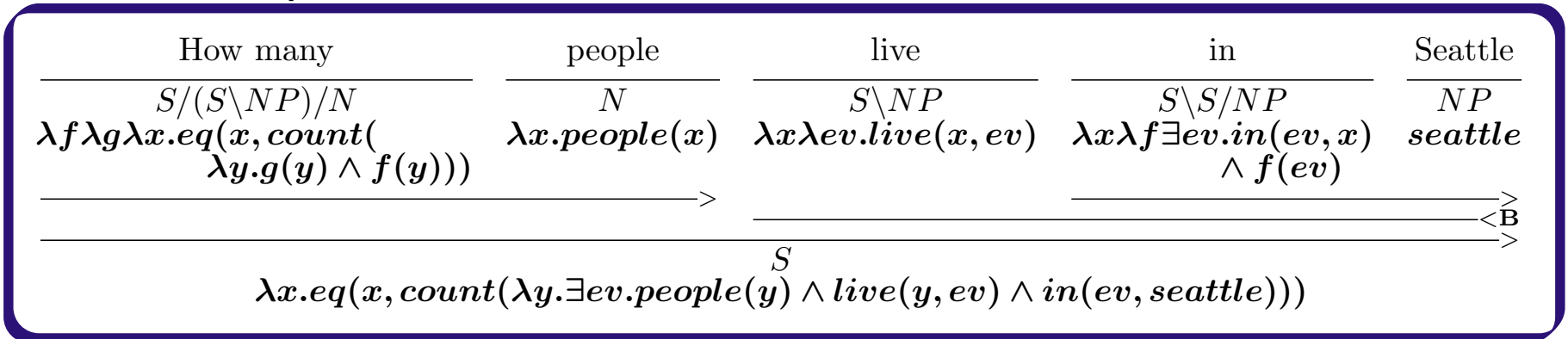
Ontology Match

Ontology Match

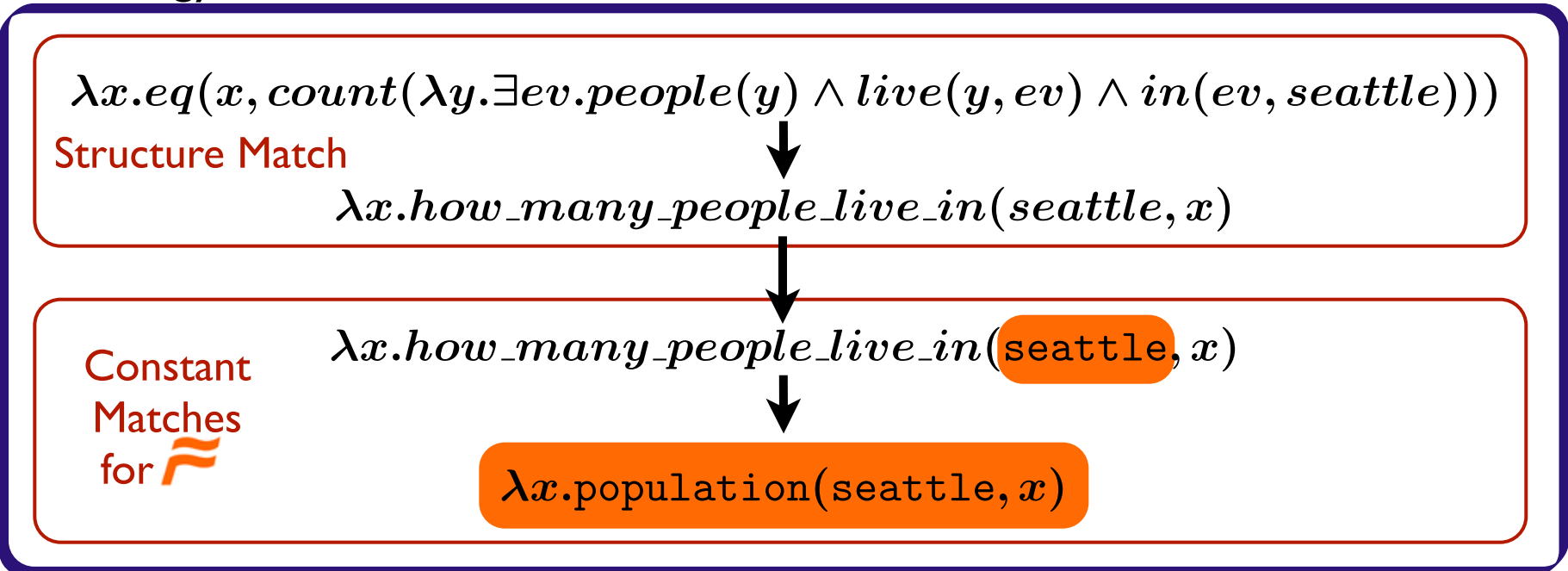


# 2 Stage Semantic Parsing

## Domain Independent Parse



## Ontology Match



# CCG Parsing

How many	people	live	in	Seattle
$S/(S \setminus NP)/N$	$N$	$S \setminus NP$	$S \setminus S/NP$	$NP$
$\lambda f \lambda g \lambda x. eq(x, count(\lambda y. f(y) \wedge g(y)))$	$\lambda x. people(x)$	$\lambda x \lambda ev. live(x, ev)$	$\lambda x \lambda f \exists ev. f(ev) \wedge in(ev, x)$	$seattle$
>			>	
$S/(S \setminus NP)$			$S \setminus S$	
$\lambda g \lambda x. eq(x, count(\lambda y. people(y) \wedge g(y)))$			$\lambda f \exists ev. f(ev) \wedge in(ev, seattle)$	
			<B	
			$S \setminus NP$	
			$\lambda s \exists ev. live(x, ev) \wedge in(ev, seattle)$	
>				
$S$				
$\lambda x. eq(x, count(\lambda y. people(y) \wedge \exists ev. live(y, ev) \wedge in(ev, seattle)))$				

## CCG Lexicon

How many  $\vdash S/(S \setminus NP)/N : \lambda f \lambda g \lambda x. eq(x, count(\lambda y. f(y) \wedge g(y)))$

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

live  $\vdash S \setminus NP : \lambda x \lambda ev. live(x, ev)$

in  $\vdash S \setminus S/NP : \lambda x \lambda f \exists ev. f(ev) \wedge in(ev, x)$

Seattle  $\vdash NP : seattle$

# CCG Parsing

How many

people

live

in

Seattle

$\lambda f$

$\lambda g$

$P$   
 $tle$

$\rightarrow$

$(le)$   
 $<B$

$\rightarrow$



Do not have lexicon for every domain

➔ Use Wiktionary for syntactic cues

➔ Parse to *underspecified* semantics

LEXICON

$in \vdash S \backslash S / NP : \lambda x \lambda f \exists ev. f(ev) \wedge in(ev, x)$

$Seattle \vdash NP : seattle$

# Domain Independent Parsing

49 domain independent lexical items:

Word		Syntax		Underspecified semantics
How many	⊢	$S/(S \setminus NP)/N$	:	$\lambda f \lambda g \lambda x. eq(x, count(\lambda y. f(y) \wedge g(y)))$
What	⊢	$S/(S \setminus NP)/N$	:	$\lambda f \lambda g \lambda x. f(x) \wedge g(x)$
most	⊢	$NP/N$	:	$\lambda f. max\_count(\lambda y. f(y))$
etc.				

# Domain Independent Parsing

How many

people

live

in

Seattle

---

$$S/(S \setminus NP)/N$$
$$\lambda f \lambda g \lambda x. eq(x, count(\lambda y. g(y) \wedge f(y)))$$



# Domain Independent Parsing

How many

people

live

in

Seattle

---

$$S/(S \setminus NP)/N$$
$$\lambda f \lambda g \lambda x. eq(x, count(\lambda y. g(y) \wedge f(y)))$$

Use Wiktionary to get part-of-speech set for words

# Domain Independent Parsing

49 domain independent lexical items:

Word	Syntax	Underspecified semantics
How many	$\vdash S/(S \setminus NP)/N$	$: \lambda f \lambda g \lambda x. eq(x, count(\lambda y. f(y) \wedge g(y)))$
What	$\vdash S/(S \setminus NP)/N$	$: \lambda f \lambda g \lambda x. f(x) \wedge g(x)$
most	$\vdash NP/N$	$: \lambda f. max\_count(\lambda y. f(y))$
etc.		

56 underspecified lexical categories:

Part-of-Speech	Syntax	Underspecified semantics
proper noun	$\vdash NP$	$: C$
noun	$\vdash N$	$: \lambda x. P(x)$
noun	$\vdash N/N$	$: \lambda f \lambda x. f(x) \wedge P(x)$
verb	$\vdash S \setminus NP$	$: \lambda \lambda ev. P(x, ev)$
verb	$\vdash S \setminus NP/NP$	$: \lambda x \lambda y \lambda ev. P(y, x, ev)$
preposition	$\vdash N \setminus N/NP$	$: \lambda f \lambda x \lambda y. P(y, x) \wedge f(y)$
preposition	$\vdash S \setminus S/NP$	$: \lambda f \lambda x \exists ev. P(ev, x) \wedge f(ev)$
etc.		

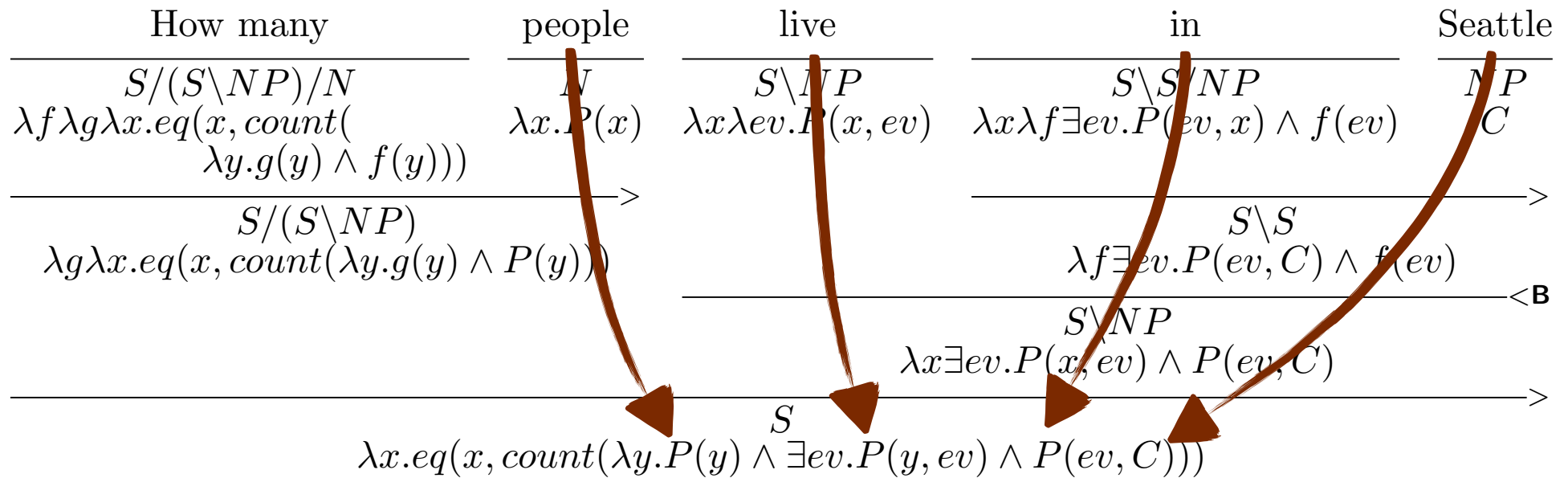
# Domain Independent Parsing

How many	people	live	in	Seattle
$S/(S \setminus NP)/N$	$N$	$S \setminus NP$	$S \setminus S/NP$	$NP$
$\lambda f \lambda g \lambda x. eq(x, count(\lambda y. g(y) \wedge f(y)))$	$\lambda x. P(x)$	$\lambda x \lambda ev. P(x, ev)$	$\lambda x \lambda f \exists ev. P(ev, x) \wedge f(ev)$	$C$

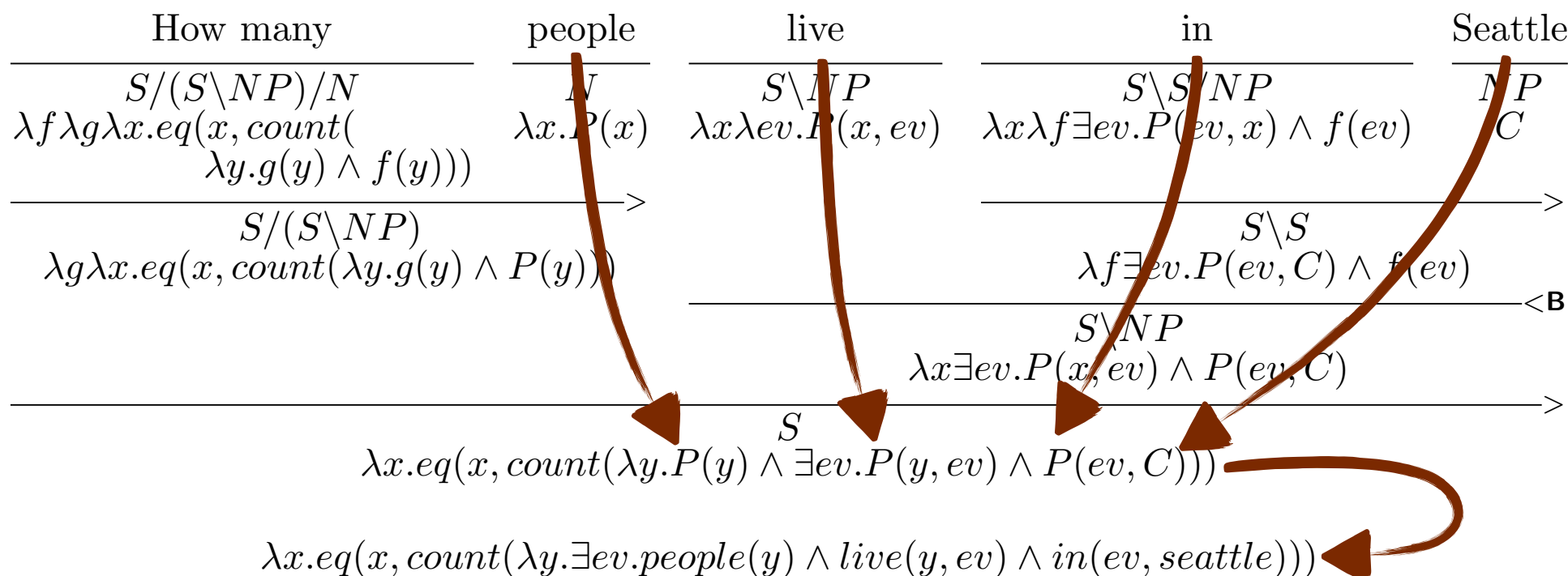
# Domain Independent Parsing

How many	people	live	in	Seattle
$S/(S \setminus NP)/N$	$N$	$S \setminus NP$	$S \setminus S/NP$	$NP$
$\lambda f \lambda g \lambda x. eq(x, count(\lambda y. g(y) \wedge f(y)))$	$\lambda x. P(x)$	$\lambda x \lambda ev. P(x, ev)$	$\lambda x \lambda f \exists ev. P(ev, x) \wedge f(ev)$	$C$
$S/(S \setminus NP)$	$\lambda g \lambda x. eq(x, count(\lambda y. g(y) \wedge P(y)))$	$S \setminus S$	$\lambda f \exists ev. P(ev, C) \wedge f(ev)$	$\langle B$
$S \setminus NP$	$\lambda x \exists ev. P(x, ev) \wedge P(ev, C)$	$S$	$\lambda x. eq(x, count(\lambda y. P(y) \wedge \exists ev. P(y, ev) \wedge P(ev, C)))$	$\rangle$

# Domain Independent Parsing



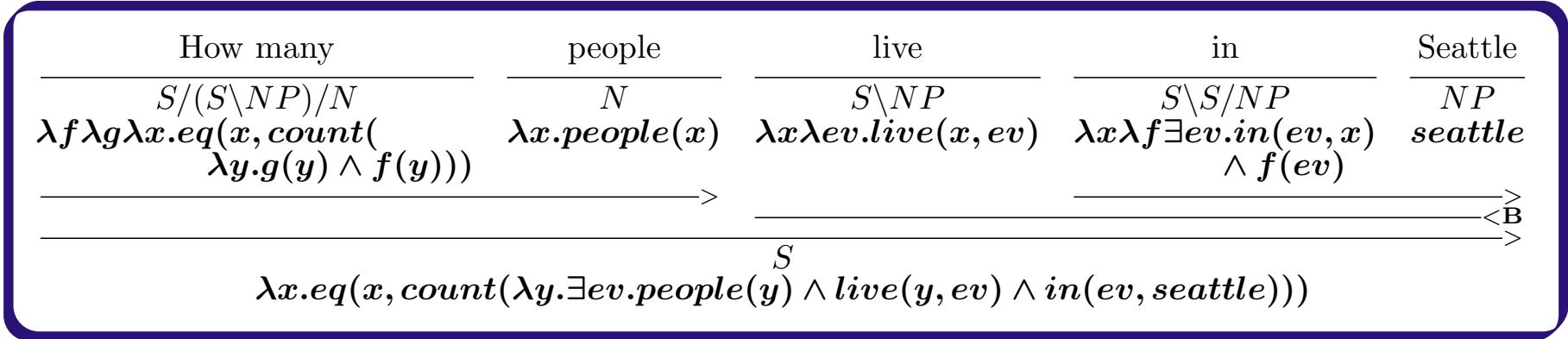
# Domain Independent Parsing



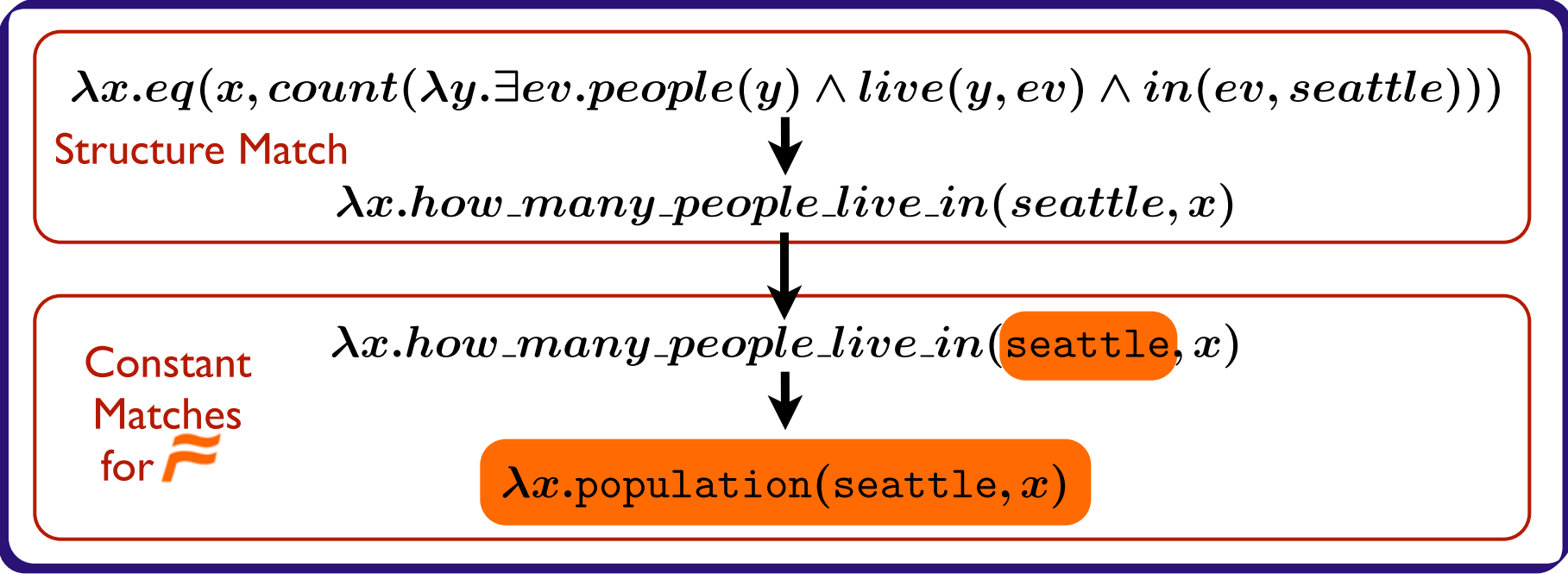
String labels signify source words, not semantic constants.

# 2 Step Semantic Parsing

## Domain Independent Parse



## Ontology Match



# 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?

$\lambda x.eq(x, count(\lambda y.people(y) \wedge \exists e.ride(y, \iota z.monorail(z) \wedge in(z, seattle), e) \wedge daily(e)))$

  $\lambda x.transit\_system/daily\_riders(seattle\_monorail, x)$



# Structural Match

1. Find subexpression with type allowed in KB

$\lambda x. eq(x, count(\lambda y. people(y) \wedge \exists e. ride(y, \iota z. monorail(z) \wedge in(z, seattle), e) \wedge daily(e)))$

2. Replace with new underspecified constant

# Structural Match

## 1. Find subexpression with type allowed in KB

$\lambda x.eq(x, count(\lambda y.people(y) \wedge \exists e.ride(y, \lambda z.monorail(z) \wedge in(z, seattle), e) \wedge daily(e)))$

entity typed  
subexpression

## 2. Replace with new underspecified constant

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

# Structural Match

1. Find subexpression with type allowed in KB

$\lambda x.eq(x, count(\lambda y.people(y) \wedge \exists e.ride(y, \iota z.monorail(z) \wedge in(z, seattle), e) \wedge daily(e)))$

integer typed  
subexpression

2. Replace with new underspecified constant

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

$\lambda x.eq(x, how\_many\_people\_ride\_daily\_the\_monorail\_in\_seattle)$

# Structural Match

1. Find subexpression with type allowed in KB

$\lambda x.eq(x, count(\lambda y.people(y) \wedge \exists e.ride(y, \iota z.monorail(z) \wedge in(z, seattle), e) \wedge daily(e)))$

integer typed  
subexpression

2. Replace with new underspecified constant

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

$\lambda x.eq(x, how\_many\_people\_ride\_daily\_the\_monorail\_in\_seattle)$

# Structural Match

## 1. Find subexpression with type allowed in KB

$\lambda x.eq(x, count(\lambda y.people(y) \wedge \exists e.ride(y, \iota z.monorail(z) \wedge in(z, seattle), e) \wedge daily(e)))$

## 2. Replace with new underspecified constant

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

$\lambda x.eq(x, how\_many\_people\_ride\_daily\_the\_monorail\_in\_seattle)$

# Structural Match

1. Find subexpression with type allowed in KB

2. Replace with new underspecified constant

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

# Structural Match

1. Find subexpression with type allowed in KB

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

2. Replace with new underspecified constant

# Structural Match

## 1. Find subexpression with type allowed in KB

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

$\langle e, \langle e, t \rangle \rangle$  typed  
subexpression

## 2. Replace with new underspecified constant

$\lambda x.eq(x, count(\lambda y.people(y) \wedge ride\_daily(y, monorail\_in\_seattle)))$



# Structural Match

## 1. Find subexpression with type allowed in KB

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

<i,<e,t>> typed  
subexpression

## 2. Replace with new underspecified constant

$\lambda x.eq(x, count(\lambda y.people(y) \wedge ride\_daily(y, monorail\_in\_seattle)))$

$\lambda x.how\_many\_people\_ride\_daily(the\_monorail\_in\_seattle, x)$

# Structural Match

## 1. Find subexpression with type allowed in KB

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

## 2. Replace with new underspecified constant

$\lambda x.eq(x, count(\lambda y.people(y) \wedge ride\_daily(y, monorail\_in\_seattle)))$

$\lambda x.how\_many\_people\_ride\_daily(the\_monorail\_in\_seattle, x)$

# Structural Match

I. Find subexpression with type allowed in KB

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

$\lambda x.how\_many\_people\_ride\_daily(the\_monorail\_in\_seattle, x)$

# 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

*$\lambda x.$ how\_many\_people\_ride\_daily(the\_monorail\_in\_seattle, x)*


  `$\lambda x.$ transit_system/daily_riders(seattle_monorail, x)`


# Constant Match

Replace constants with constants from KB

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

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



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

# Constant Match

Replace constants with constants from KB

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

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

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

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



# 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

How many	people	live	in	Seattle
$S/(S \setminus NP)/N$	$N$	$S \setminus NP$	$S \setminus S/NP$	$NP$
$\lambda f \lambda g \lambda x. eq(x, count(\lambda y. g(y) \wedge f(y)))$	$\lambda x. people(x)$	$\lambda x \lambda ev. live(x, ev)$	$\lambda x \lambda f \exists ev. in(ev, x) \wedge f(ev)$	$seattle$
>		>		
		<B		
$\lambda x. eq(x, count(\lambda y. \exists ev. people(y) \wedge live(y, ev) \wedge in(ev, seattle)))$				

## Ontology Match

d

$\lambda x. eq(x, count(\lambda y. \exists ev. people(y) \wedge live(y, ev) \wedge in(ev, seattle)))$

Structure Match

$\lambda x. how\_many\_people\_live\_in(seattle, x)$

Constant Matches for 

$\lambda x. how\_many\_people\_live\_in(seattle, x)$

$\lambda x. population(seattle, x)$

# Scoring Derivations

Derivations  $d$  are scored using a linear model:

$$\text{score}(d) = \phi(d)\theta$$

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?

$\lambda x. \text{when\_first\_air}(\text{prairie\_home\_companion}, x)$

  $\lambda x. \text{radio\_program.} \text{first\_broadcast}(\text{prairie\_home\_companion}, x)$



How high is Niagara Falls?

$\lambda x. \text{high}(\text{niagara\_falls}, x)$

  $\lambda x. \text{location.geocode.} \text{elevation}(\text{niagara\_falls}, x)$



- 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*)

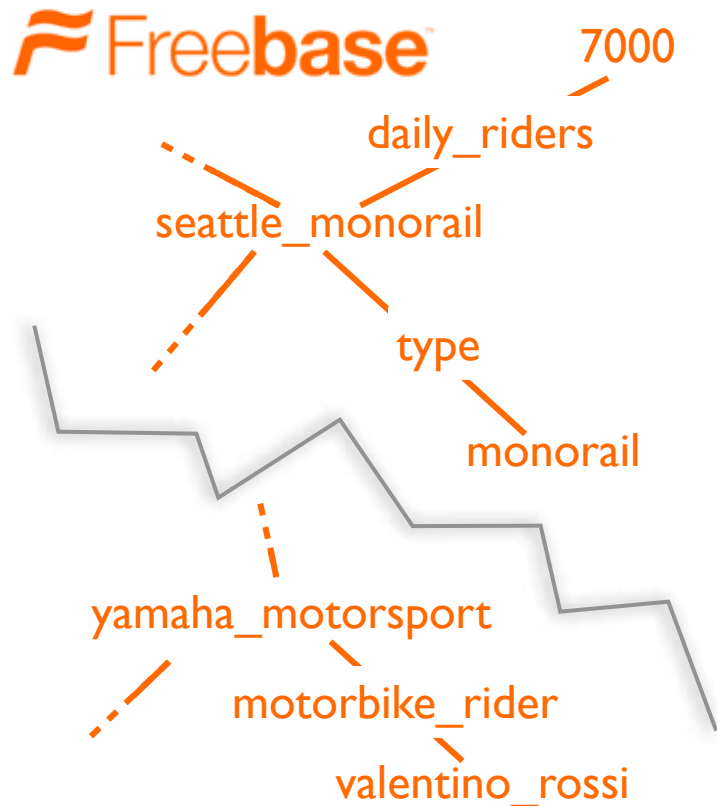
1. Being elevated in position or status, a state of being above many things. [quotations ▼]

$\lambda x. high(niagara\_falls, x)$

$\approx \lambda x. location.geocode.elevation(niagara\_falls, x)$

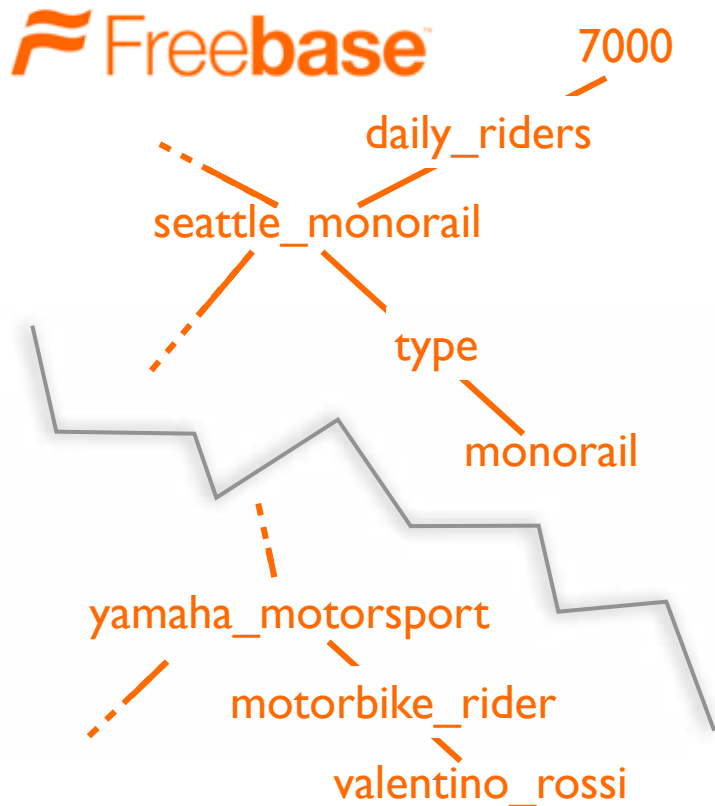
- Exact string match.
- Stemmed string match.
- Synonym match.
- Wiktionary gloss overlap.

# Knowledge Base Features



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

# Knowledge Base Features



## Predicted Logical Form

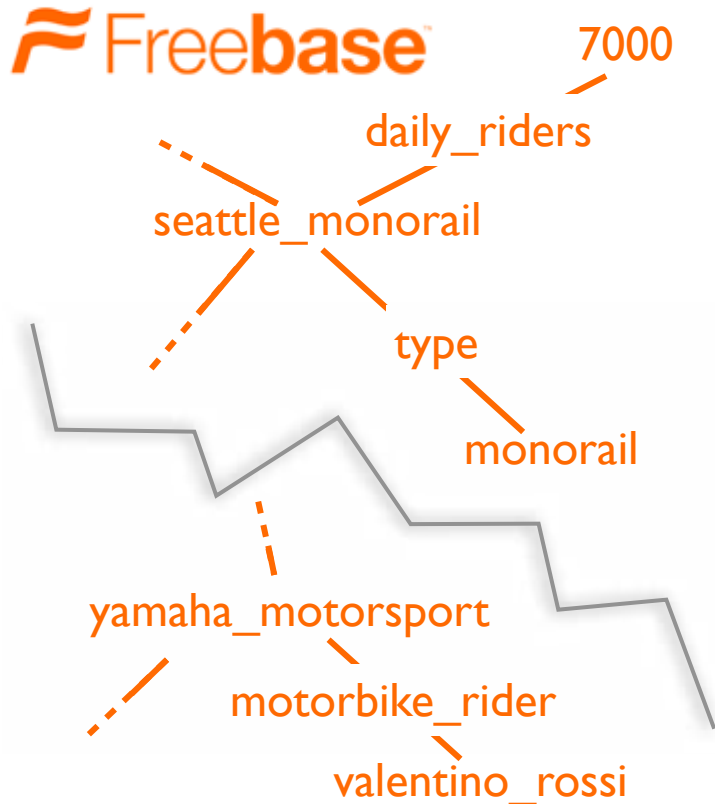
$\lambda x.\text{daily\_riders}(\text{seattle\_monorail}, x)$

$\lambda x.\text{motorbike\_rider}(\text{seattle\_monorail}, x)$



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

# Knowledge Base Features



## Predicted Logical Form

$\lambda x \lambda y. \text{motorbike\_rider}(x, y) \wedge \text{monorail}(x)$

$\lambda x \lambda y. \text{daily\_riders}(x, y) \wedge \text{monorail}(x)$



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

How many	people	live	in	Seattle
$S/(S \setminus NP)/N$	$N$	$S \setminus NP$	$S \setminus S/NP$	$NP$
$\lambda f \lambda g \lambda x. eq(x, count(\lambda y. g(y) \wedge f(y)))$	$\lambda x. people(x)$	$\lambda x \lambda ev. live(x, ev)$	$\lambda x \lambda f \exists ev. in(ev, x) \wedge f(ev)$	$seattle$
>				
<B				
$S$				
$\lambda x. eq(x, count(\lambda y. \exists ev. people(y) \wedge live(y, ev) \wedge in(ev, seattle)))$				

## Ontology Match

d

$\lambda x. eq(x, count(\lambda y. \exists ev. people(y) \wedge live(y, ev) \wedge in(ev, seattle)))$

$\lambda x. eq(x, count(\lambda y. \exists ev. people(y) \wedge live(y, ev) \wedge in(ev, seattle)))$

$\lambda x. how\_many\_people\_live\_in(seattle, x)$

$\lambda x. population(seattle, x)$

# 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  $\{(x_i, a_i) : i = 1, \dots, n\}$

Knowledge Base, Wiktionary, Underspecified Lexicon

## Algorithm

For  $i = 1, \dots, n$  :

$C \leftarrow$  Max scoring correct parses of  $x_i$

$W \leftarrow$  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

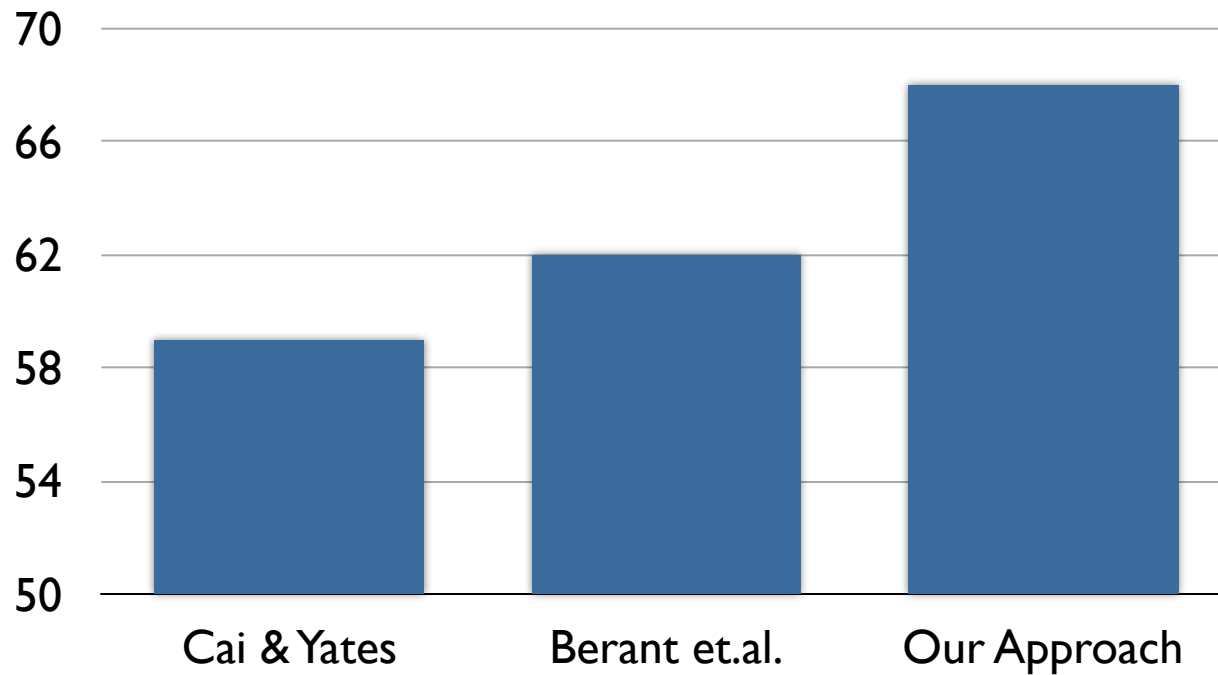
## Learning CCG from Labelled Logical Forms:

-  ▸ Kwiatkowski et.al. 2011 (FUBL)
-  ▸ Cai & Yates 2013



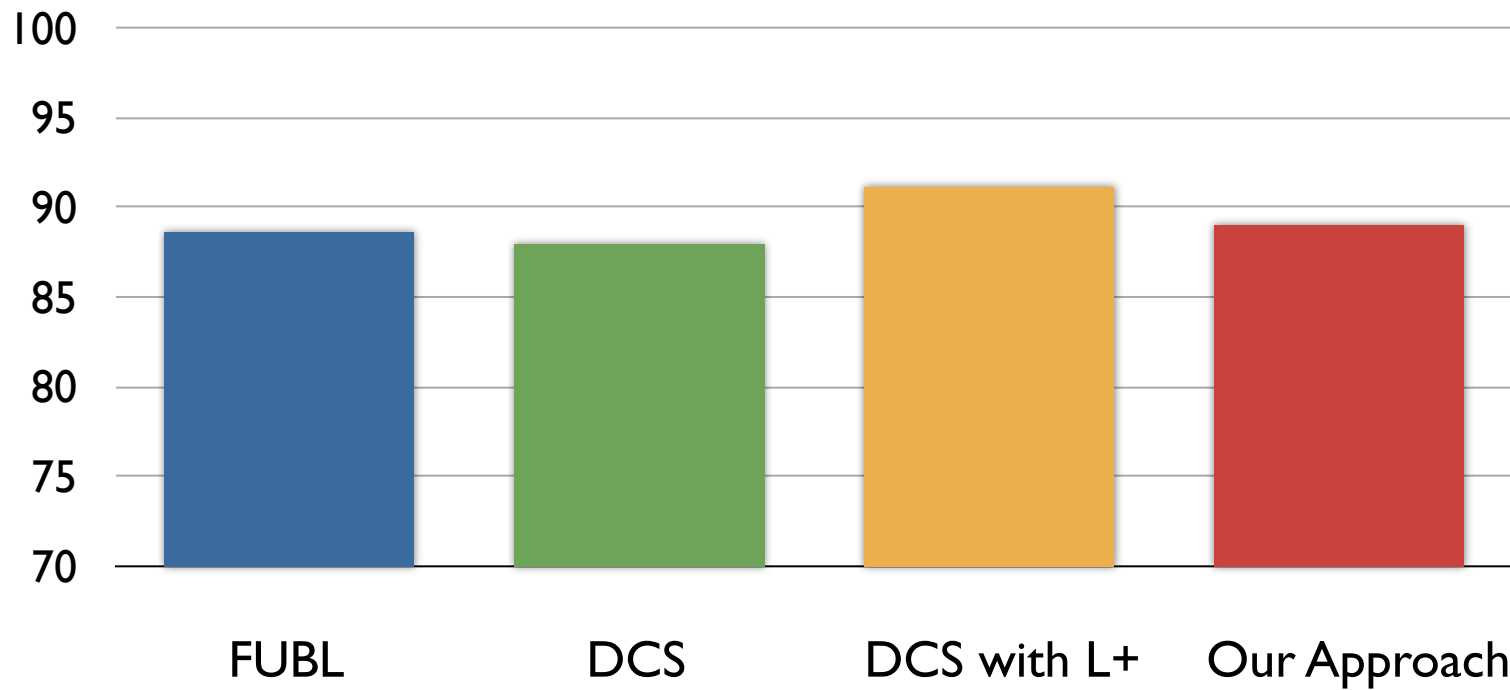
# Experiments

## Freebase 917



# Experiments

## GeoQuery





# Example Parses

Who is the CEO of Save-A-Lot?

$\lambda x.\text{person}(x) \wedge \exists y.\text{organization}(y, \text{savealot}) \wedge$




$\text{board\_member.leader\_of}(x, y) \wedge \text{leadership.role}(y, \text{ceo})$

# Example Errors

How many children does Jerry Seinfeld have?

Target:

  $\lambda x.\text{eq}(x, \text{count}(\lambda y.\text{person.children}(\text{jerry\_seinfeld}, y)))$

Prediction:

  $\lambda x.\text{eq}(x, \text{count}(\lambda y.\text{person.children}(y, \text{jerry\_seinfeld})))$

# Example Errors

What programming languages were used for AOL instant messenger?

Target:



```
λx.languages_used(aol_instant_messenger, x)
```

Prediction:



```
λx.languages_used(aol_instant_messenger, x)
```


```
∧programming_language(x)
```

# Future Work

## Information Extraction:

**Alan Turing** was a British mathematician, [logician](#), [cryptanalyst](#), and [computer scientist](#).

```
nationality(AT, UK) ^ notable_for(AT, mathematician)
^ profession(AT, logic) ^ research(AT, cryptanalysis)
^ notable_type(AT, compsci)
```

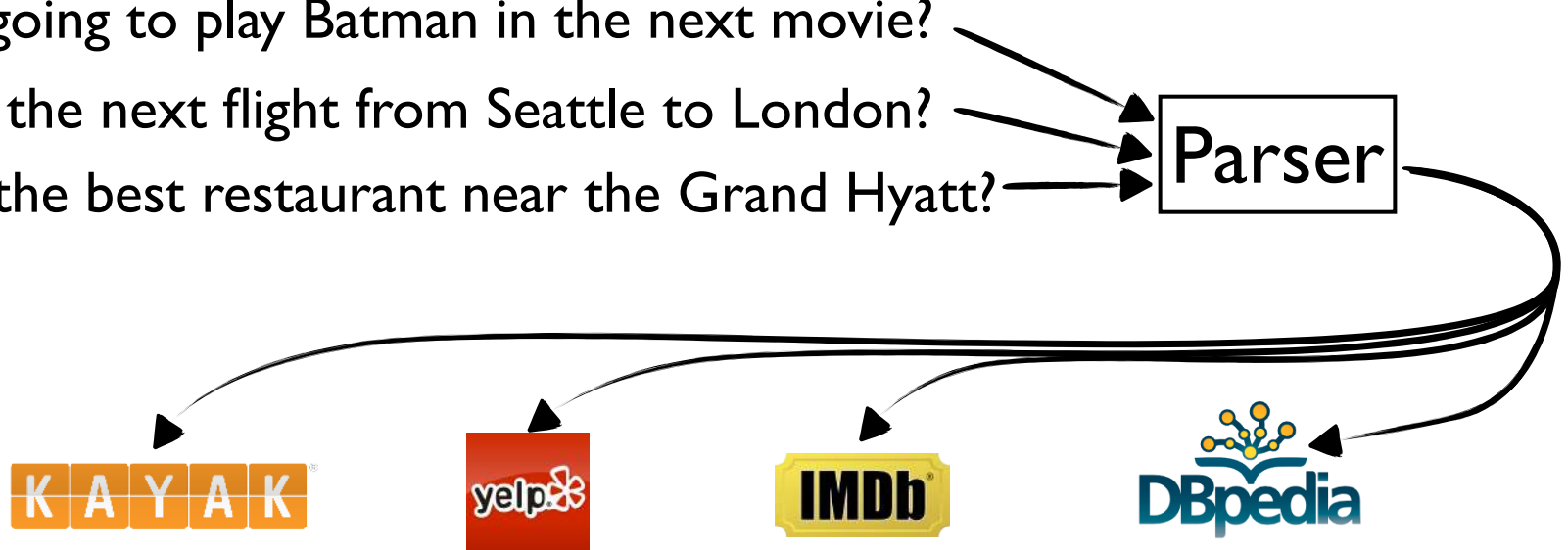


## Multiple databases:

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?



# Questions

?