Semantic Parsing with Combinatory Categorial Grammars

Yoav Artzi
University of Washington

Based on ACL 2013 Tutorial
With Nicholas FitzGerald and Luke Zettlemoyer
Original tutorial slides available at http://yoavartzi.com/
Language to Meaning

More informative
Language to Meaning

Information Extraction

Recover information about pre-specified relations and entities

Example Task

Relation Extraction

is_a(Obama, President)

The New York Times
OBAMA
Racial barrier falls in decisive victory

Democrats in Congress strengthen grip
Language to Meaning

Broad-coverage Semantics

Focus on specific phenomena (e.g., verb-argument matching)

More informative

Example Task

Summarization

Obama wins election. Big party in Chicago. Romney a bit down, asks for some tea.
Language to Meaning

Example Task

Database Query

What states border Texas?

Oklahoma
New Mexico
Arkansas
Louisiana
Language to Meaning

Example Task

Instructing a Robot

at the chair,

turn right
Language to Meaning

Complete meaning is sufficient to complete the task

- Convert to database query to get the answer
- Allow a robot to do planning
at the chair, move forward three steps past the sofa

\( \lambda a. \text{pre}(a, \mu x. \text{chair}(x)) \land \text{move}(a) \land \text{len}(a, 3) \land \text{dir}(a, \text{forward}) \land \text{past}(a, \mu y. \text{sofa}(y)) \)
Language to Meaning

\(\lambda a.\) pre\((a, \forall x.\) chair\((x)\)) \land move\((a)\) \land len\((a, 3)\) \land dir\((a, forward)\) \land past\((a, \forall y. sofa(y))\)

at the chair, move forward three steps past the sofa
at the chair, move forward three steps past the sofa

\[ \lambda a. \text{pre}(a, \forall x. \text{chair}(x)) \land \text{move}(a) \land \text{len}(a, 3) \land \text{dir}(a, \text{forward}) \land \text{past}(a, \forall y. \text{sofa}(y)) \]

\[ f : \text{sentence} \rightarrow \text{logical form} \]
Language to Meaning

at the chair, move forward three steps past the sofa

\[ f \colon \text{sentence} \rightarrow \text{logical form} \]
Central Problems

- Parsing
- Learning
- Modeling
Parsing Choices

- Grammar formalism
- Inference procedure

Inductive Logic Programming [Zelle and Mooney 1996]
SCFG [Wong and Mooney 2006]
CCG + CKY [Zettlemoyer and Collins 2005]
Constrained Optimization + ILP [Clarke et al. 2010]
DCS + Projective dependency parsing [Liang et al. 2011]
LWFG [Muresan 2011]
Learning

• What kind of supervision is available?
• Mostly using latent variable methods

Annotated parse trees [Miller et al. 1994]
Sentence-LF pairs [Zettlemoyer and Collins 2005]
Question-answer pairs [Clarke et al. 2010]
Instruction-demonstration pairs [Chen and Mooney 2011]
Conversation logs [Artzi and Zettlemoyer 2011]
Visual sensors [Matuszek et al. 2012a]
Semantic Modeling

• What logical language to use?

• How to model meaning?

Variable free logic [Zelle and Mooney 1996; Wong and Mooney 2006]
High-order logic [Zettlemoyer and Collins 2005]
Relational algebra [Liang et al. 2011]
Graphical models [Tellex et al. 2011]
Today

Parsing
Combimatory Categorial Grammars

Learning
Unified learning algorithm

Modeling
Best practices for semantics design
• Lambda calculus
• Parsing with Combinatory Categorial Grammars
• Linear CCGs
• Factored lexicons
• Structured perceptron
• A unified learning algorithm
• Supervised learning
• Weak supervision Online
• Semantic modeling for:
  - Querying databases
    - Referring to physical objects
    - Executing instructions
UWV SPF

Open source semantic parsing framework

http://yoavartzi.com/spf

Semantic Parser
Flexible High-Order Logic Representation
Learning Algorithms

Includes ready-to-run examples

[Artzi and Zettlemoyer 2013a]
• Lambda calculus
• Parsing with Combinatory Categorial Grammars
• Linear CCGs

Factored lexicons Online
Lambda Calculus

• Formal system to express computation
• Allows high-order functions

\[ \lambda a.\text{move}(a) \land \text{dir}(a, \text{LEFT}) \land \text{to}(a, \forall y.\text{chair}(y)) \land \text{pass}(a, \forall y.\text{sofa}(y) \land \text{intersect}(\forall z.\text{intersection}(z, y))) \]

[Church 1932]
Lambda Calculus

Base Cases

- Logical constant
- Variable
- Literal
- Lambda term
Lambda Calculus
Logical Constants

- Represent objects in the world

NYC, CA, RAINIER, LEFT, ...
located_in, depart_date, ...
Lambda Calculus

Variables

• Abstract over objects in the world
• Exact value not pre-determined

\( x, y, z, \ldots \)
Lambda Calculus

Literals

- Represent function application

\[ \text{city}(AUSTIN) \]
\[ \text{located_in}(AUSTIN, TEXAS) \]
Lambda Calculus

Literals

- Represent function application

\[ \text{city}(AUSTIN) \]

\[ \text{located}_{in}(AUSTIN, TEXAS) \]

Predicate | Arguments
---|---
Logical expression | List of logical expressions
Lambda Calculus

Lambda Terms

• Bind/scope a variable
• Repeat to bind multiple variables

\[ \lambda x. \text{city}(x) \]
\[ \lambda x. \lambda y. \text{located}_\text{in}(x, y) \]
Lambda Calculus

Lambda Terms

- Bind/scope a variable
- Repeat to bind multiple variables

\[ \lambda x.\text{city}(x) \]

\[ \lambda x.\lambda y.\text{located_in}(x, y) \]
Lambda Calculus

Quantifiers?

- Higher order constants
- No need for any special mechanics
- Can represent all of first order logic

\[ \forall (\lambda x. \text{big}(x) \land \text{apple}(x)) \]
\[ \neg (\exists (\lambda x. \text{lovely}(x))) \]
\[ \iota (\lambda x. \text{beautiful}(x) \land \text{grammar}(x)) \]
Lambda Calculus

Syntactic Sugar

\( \wedge (A, \wedge (B, C)) \iff A \wedge B \wedge C \)

\( \vee (A, \vee (B, C)) \iff A \vee B \vee C \)

\( \neg (A) \iff \neg A \)

\( Q(\lambda x. f(x)) \iff Qx. f(x) \)

for \( Q \in \{ \iota, A, \exists, \forall \} \)
\[
\lambda x. \text{flight}(x) \land \text{to}(x, \text{move}) \\
\lambda x. \text{flight}(x) \land \text{to}(x, \text{NYC}) \\
\lambda x. \text{NYC}(x) \land x(\text{to, move})
\]
✗ \( \lambda x. \text{flight}(x) \land \text{to}(x, \text{move}) \)
✓ \( \lambda x. \text{flight}(x) \land \text{to}(x, \text{NYC}) \)
✗ \( \lambda x. \text{NYC}(x) \land x(\text{to}, \text{move}) \)
Simply Typed Lambda Calculus

- Like lambda calculus
- But, typed

\[ \lambda x. \text{flight}(x) \land \text{to}(x, \text{move}) \]
\[ \checkmark \lambda x. \text{flight}(x) \land \text{to}(x, \text{NYC}) \]
\[ \times \lambda x. \text{NYC}(x) \land x(\text{to, move}) \]

[Church 1940]
Lambda Calculus

Typing

• Simple types
• Complex types

\(<e, t>\)

\(<\langle e, t>, e>\)
Lambda Calculus

Typing

- Simple types
- Complex types

\[ \langle e, t \rangle \]

\[ \langle \langle e, t \rangle, e \rangle \]

Domain

Range

Type constructor

Truth-value

Entity
Lambda Calculus

Typing

- Simple types
- Complex types
- Hierarchical typing system

\[ \langle e, t \rangle \]

\[ \langle \langle e, t \rangle, e \rangle \]

Type constructor

Domain

Range
Lambda Calculus

Typing

- Simple types
- Complex types

Hierarchical typing system

- Type constructor
- Domain
- Range

\[ < e, t > \]
Simply Typed Lambda Calculus

\[ \lambda a. move(a) \land dir(a, LEFT) \land to(a, \nu y. chair(y)) \land pass(a, \forall y. sofa(y) \land intersect(\forall z. intersection(z), y)) \]

Type information usually omitted
Capturing Meaning with Lambda Calculus

<table>
<thead>
<tr>
<th>State</th>
<th>Abbr.</th>
<th>Capital</th>
<th>Pop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
<td>3.9</td>
<td></td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
<td>2.7</td>
<td></td>
</tr>
</tbody>
</table>

[Border]

<table>
<thead>
<tr>
<th>State1</th>
<th>State2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>OR</td>
</tr>
<tr>
<td>WA</td>
<td>ID</td>
</tr>
<tr>
<td>CA</td>
<td>OR</td>
</tr>
<tr>
<td>CA</td>
<td>NV</td>
</tr>
</tbody>
</table>

[Mountains]

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
</tr>
<tr>
<td>Wrangel</td>
<td>AK</td>
</tr>
<tr>
<td>Sill</td>
<td>CA</td>
</tr>
</tbody>
</table>

Show me mountains in states bordering Texas

[Zettlemoyer and Collins 2005]
Capturing Meaning with Lambda Calculus

**System**: how can I help you?

**User**: I’d like to fly to New York

**System**: flying to New York. Leaving what city?

**User**: from Boston on June seven with American Airlines

**System**: flying to New York. What date would you like to depart Boston?

**User**: June seventh

**System**: Do you have a preferred airline?

**User**: American Airlines

**System**: o.k. Leaving Boston to New York on June seventh flying with American Airlines. Where would you like to go to next?

**User**: back to Boston on June tenth

[Conversatıon Continues]
Capturing Meaning with Lambda Calculus

\[ \lambda a. \text{move}(a) \land \text{to}(a, \ldots) \]

go to the chair and turn right

[Artzi and Zettlemoyer 2013b]
Capturing Meaning with Lambda Calculus

- Flexible representation
- Can capture full complexity of natural language

More on modeling meaning online and later today
Constructing Lambda Calculus Expressions

at the chair, move forward three steps past the sofa

\[ \lambda a. \text{pre}(a, \forall x. \text{chair}(x)) \land \text{move}(a) \land \text{len}(a, 3) \land \text{dir}(a, \text{forward}) \land \text{past}(a, \forall y. \text{sofa}(y)) \]
Combinatory Categorial Grammars

\[
\begin{align*}
\text{CCG} & \quad \text{is} \quad \text{fun} \\
NP & \quad S \smallsetminus NP/ADJ \\
CCG & \quad \lambda f.\lambda x.f(x) \\
& \quad \lambda x.\text{fun}(x) \\
& \quad \rightarrow \\
S \smallsetminus NP & \quad \lambda x.\text{fun}(x) \\
& \quad \leftarrow \\
& \quad \text{fun}(\text{CCG})
\end{align*}
\]

[Steedman 1996, 2000]
Combinatory Categorial Grammars

- Categorial formalism
- Transparent interface between syntax and semantics
- Designed with computation in mind
- Part of a class of mildly context sensitive formalisms (e.g., TAG, HG, LIG) [Joshi et al. 1990]
CCG Categories

\[ ADJ : \lambda x. \text{fun}(x) \]

- Basic building block
- Capture syntactic and semantic information jointly
CCG Categories

- Basic building block
- Capture syntactic and semantic information jointly
### CCG Categories

**Syntax**

- **ADJ**: $\lambda x.\text{fun}(x)$
- $(S\backslash NP) / \text{ADJ} : \lambda f.\lambda x.f(x)$
- **NP**: $\text{CCG}$

- Primitive symbols: N, S, NP, ADJ and PP
- Syntactic combination operator (/,\)
- Slashes specify argument order and direction
CCG Categories

\[ ADJ : \lambda x. \text{fun}(x) \]

\[ (S \backslash NP)/\text{ADJ} : \lambda f. \lambda x. f(x) \]

\[ NP : CCG \]

- \( \lambda \)-calculus expression
- Syntactic type maps to semantic type
CCG Lexical Entries

\[ \text{fun} \vdash \text{ADJ} : \lambda x. \text{fun}(x) \]

- Pair words and phrases with meaning
- Meaning captured by a CCG category
CCG Lexical Entries

- Pair words and phrases with meaning
- Meaning captured by a CCG category
CCG Lexicons

fun ⊨ ADJ : λx.\text{fun}(x)

is ⊨ (S\setminus NP)/ADJ : λf.λx.f(x)

CCG ⊨ NP : CCG

• Pair words and phrases with meaning

• Meaning captured by a CCG category
## Between CCGs and CFGs

<table>
<thead>
<tr>
<th></th>
<th>CFGs</th>
<th>CCGs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combination operations</td>
<td>Many</td>
<td>Few</td>
</tr>
<tr>
<td>Parse tree nodes</td>
<td>Non-terminals</td>
<td>Categories</td>
</tr>
<tr>
<td>Syntactic symbols</td>
<td>Few dozen</td>
<td>Handful, but can combine</td>
</tr>
<tr>
<td>Paired with words</td>
<td>POS tags</td>
<td>Categories</td>
</tr>
</tbody>
</table>
Parsing with CCGs

Use lexicon to match words and phrases with their categories
CCG Operations

- Small set of operators
- Input: 1-2 CCG categories
- Output: A single CCG category
- Operate on syntax semantics together
- Mirror natural logic operations
CCG Operations
Application

\[ B : g \quad A \backslash B : f \Rightarrow A : f(g) \quad (<) \]
\[ A / B : f \quad B : g \Rightarrow A : f(g) \quad (>) \]

- Equivalent to function application
- Two directions: forward and backward
  - Determined by slash direction
CCG Operations
Application

Argument | Function | Result
--- | --- | ---
$B : g$ | $A \backslash B : f$ | $A : f(g)$

$A / B : f \quad B : g \Rightarrow A : f(g)$

- Equivalent to function application
- Two directions: forward and backward
  - Determined by slash direction
Parsing with CCGs

Use lexicon to match words and phrases with their categories
Parsing with CCGs

CCG

NP

CCG

is

fun

$S \backslash NP/ADJ$

$\lambda f. \lambda x.f(x)$

$ADJ$

$\lambda x. fun(x)$

$S \backslash NP$

$\lambda x. fun(x)$

Combine categories using operators

$A/B : f \quad B : g \Rightarrow A : f(g)$ (>)
Parsing with CCGs

Combine categories using operators

\[ B : g \quad A \setminus B : f \Rightarrow A : f(g) \quad (\langle \rangle) \]
Parsing with CCGs

Composed adjectives

square blue or round yellow pillow

Non-standard coordination
CCG Operations
Composition

\[
\begin{align*}
A/B : f & \quad B/C : g \Rightarrow A/C : \lambda x. f(g(x)) \quad (> B) \\
B\setminus C : g & \quad A\setminus B : f \Rightarrow A\setminus C : \lambda x. f(g(x)) \quad (< B)
\end{align*}
\]

- Equivalent to function composition*
- Two directions: forward and backward

* Formal definition of logical composition in supplementary slides
CCG Operations
Composition

$\begin{align*}
f & : A/B \\
g & : B/C \\
f \circ g & : A/C \\
\end{align*}$

$\begin{align*}
A/B : f & \Rightarrow A/C : \lambda x. f(g(x)) (\geq B) \\
B/C : g & \Rightarrow B \setminus C : g \\
A \setminus B : f & \Rightarrow A \setminus C : \lambda x. f(g(x)) (\leq B)
\end{align*}$

- Equivalent to function composition*
- Two directions: forward and backward

* Formal definition of logical composition in supplementary slides
CCG Operations
Type Shifting

\[ ADJ : \lambda x.g(x) \Rightarrow N/N : \lambda f.\lambda x.f(x) \land g(x) \]
\[ PP : \lambda x.g(x) \Rightarrow N\setminus N : \lambda f.\lambda x.f(x) \land g(x) \]
\[ AP : \lambda e.g(e) \Rightarrow S\setminus S : \lambda f.\lambda e.f(e) \land g(e) \]
\[ AP : \lambda e.g(e) \Rightarrow S/S : \lambda f.\lambda e.f(e) \land g(e) \]

- Category-specific unary operations
- Modify category type to take an argument
- Helps in keeping a compact lexicon
## CCG Operations

### Type Shifting

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ADJ : \lambda x.g(x)$</td>
<td>$N/N : \lambda f.\lambda x.f(x) \land g(x)$</td>
</tr>
<tr>
<td>$PP : \lambda x.g(x)$</td>
<td>$N\backslash N : \lambda f.\lambda x.f(x) \land g(x)$</td>
</tr>
<tr>
<td>$AP : \lambda e.g(e)$</td>
<td>$S\backslash S : \lambda f.\lambda e.f(e) \land g(e)$</td>
</tr>
<tr>
<td>$AP : \lambda e.g(e)$</td>
<td>$S/S : \lambda f.\lambda e.f(e) \land g(e)$</td>
</tr>
</tbody>
</table>

- Category-specific unary operations
- Modify category type to take an argument
- Helps in keeping a compact lexicon
CCG Operations
Type Shifting

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ADJ : \lambda x.g(x))</td>
<td>(N/N : \lambda f.\lambda x.f(x) \land g(x))</td>
</tr>
<tr>
<td>(PP : \lambda x.g(x))</td>
<td>(N \setminus N : \lambda f.\lambda x.f(x) \land g(x))</td>
</tr>
<tr>
<td>(AP : \lambda e.g(e))</td>
<td>(S \setminus S : \lambda f.\lambda e.f(e) \land g(e))</td>
</tr>
<tr>
<td>(AP : \lambda e.g(e))</td>
<td>(\boxed{S/S} : \lambda f.\lambda e.f(e) \land g(e))</td>
</tr>
</tbody>
</table>

- Category-specific unary operations
- Modify category type to take an argument
- Helps in keeping a compact lexicon
CCG Operations
Coordination

\[ \text{and } \vdash C : \text{conj} \]

\[ \text{or } \vdash C : \text{disj} \]

- Coordination is special cased
  - Specific rules perform coordination
  - Coordinating operators are marked with special lexical entries
Parsing with CCGs

square blue or round yellow pillow
## Parsing with CCGs

<table>
<thead>
<tr>
<th>square</th>
<th>blue</th>
<th>or</th>
<th>round</th>
<th>yellow</th>
<th>pillow</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{ADJ}$ $\lambda x.\text{square}(x)$</td>
<td>$\text{ADJ} \lambda x.\text{blue}(x)$</td>
<td>$\text{C} \lambda x.\text{round}(x)$</td>
<td>$\text{ADJ} \lambda x.\text{yellow}(x)$</td>
<td>$\text{N} \lambda x.\text{pillow}(x)$</td>
<td></td>
</tr>
</tbody>
</table>

Use lexicon to match words and phrases with their categories
### Parsing with CCGs

<table>
<thead>
<tr>
<th>square</th>
<th>blue</th>
<th>or</th>
<th>round</th>
<th>yellow</th>
<th>pillow</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{ADJ}$ $\lambda x.\text{square}(x)$</td>
<td>$\text{ADJ}$ $\lambda x.\text{blue}(x)$</td>
<td>$C$ $\text{disj}$</td>
<td>$\text{ADJ}$ $\lambda x.\text{round}(x)$</td>
<td>$\text{ADJ}$ $\lambda x.\text{yellow}(x)$</td>
<td>$\text{N}$ $\lambda x.\text{pillow}(x)$</td>
</tr>
<tr>
<td>$\text{N/N}$ $\lambda f.\lambda x.f(x) \land \text{square}(x)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Shift adjectives to combine**

\[ \text{ADJ} : \lambda x. g(x) \Rightarrow \text{N/N} : \lambda f. \lambda x. f(x) \land g(x) \]
 Parsing with CCGs

<table>
<thead>
<tr>
<th></th>
<th>square</th>
<th>blue</th>
<th>or disj</th>
<th>round</th>
<th>yellow</th>
<th>pillow</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJ</td>
<td>$\lambda x.\text{square}(x)$</td>
<td>$\lambda x.\text{blue}(x)$</td>
<td>C</td>
<td>$\lambda x.\text{round}(x)$</td>
<td>$\lambda x.\text{yellow}(x)$</td>
<td>$\lambda x.\text{pillow}(x)$</td>
</tr>
<tr>
<td>N/N</td>
<td>$\lambda f.\lambda x. f(x) \land \text{square}(x)$</td>
<td>$\lambda f.\lambda x. f(x) \land \text{blue}(x)$</td>
<td>disj</td>
<td>$\lambda f.\lambda x. f(x) \land \text{round}(x)$</td>
<td>$\lambda f.\lambda x. f(x) \land \text{yellow}(x)$</td>
<td></td>
</tr>
</tbody>
</table>

**Shift adjectives to combine**

$$ADJ : \lambda x. g(x) \Rightarrow N/N : \lambda f.\lambda x. f(x) \land g(x)$$
## Parsing with CCGs

### Compose pairs of adjectives

\[
A/B : f \quad B/C : g \Rightarrow A/C : \lambda x. f(g(x)) \quad (> B)
\]
Parsing with CCGs

Coordinate composed adjectives

- square
  - $\lambda x. \text{square}(x)$
  - $\lambda f. \lambda x. f(x) \land \text{square}(x)$
- blue
  - $\lambda x. \text{blue}(x)$
  - $\lambda f. \lambda x. f(x) \land \text{blue}(x)$
- or
  - $\text{disj}$
- round
  - $\lambda x. \text{round}(x)$
  - $\lambda f. \lambda x. f(x) \land \text{round}(x)$
- yellow
  - $\lambda x. \text{yellow}(x)$
  - $\lambda f. \lambda x. f(x) \land \text{yellow}(x)$
- pillow
  - $\lambda x. \text{pillow}(x)$
  - $\lambda f. \lambda x. f(x)$

$\lambda f. \lambda x. f(x) \land (\text{square}(x) \land \text{blue}(x)) \lor (\text{round}(x) \land \text{yellow}(x))$
## Parsing with CCGs

<table>
<thead>
<tr>
<th>square</th>
<th>blue</th>
<th>or</th>
<th>round</th>
<th>yellow</th>
<th>pillow</th>
</tr>
</thead>
<tbody>
<tr>
<td>`ADJ</td>
<td>`ADJ</td>
<td>`C</td>
<td>`ADJ</td>
<td>`ADJ</td>
<td>`N</td>
</tr>
<tr>
<td><code>lx.square(x)</code></td>
<td><code>lx.blue(x)</code></td>
<td><code>disj</code></td>
<td><code>lx.round(x)</code></td>
<td><code>lx.yellow(x)</code></td>
<td><code>lx.pillow(x)</code></td>
</tr>
<tr>
<td><code>N/N</code></td>
<td><code>N/N</code></td>
<td><code>disj</code></td>
<td><code>N/N</code></td>
<td><code>N/N</code></td>
<td><code>N</code></td>
</tr>
<tr>
<td><code>N/N</code></td>
<td><code>N/N</code></td>
<td>(\lambda f.\lambda x.f(x) \land square(x))</td>
<td><code>N/N</code></td>
<td><code>N/N</code></td>
<td>(\lambda f.\lambda x.f(x) \land yellow(x))</td>
</tr>
</tbody>
</table>

### Apply coordinated adjectives to noun

\[
A/B : f \quad B : g \Rightarrow A : f(g) \quad (>)
\]
Parsing with CCGs

Lexical Ambiguity + Many parsing decisions \rightarrow Many potential trees and LFs
Weighted Linear CCGs

- Given a weighted linear model:
  - CCG lexicon $\Lambda$
  - Feature function $f : X \times Y \rightarrow \mathbb{R}^m$
  - Weights $w \in \mathbb{R}^m$

- The best parse is:
  $$y^* = \arg \max_y w \cdot f(x, y)$$

- We consider all possible parses $y$ for sentence $x$ given the lexicon $\Lambda$
Parsing Algorithms

• Syntax-only CCG parsing has polynomial time CKY-style algorithms

• Parsing with semantics requires entire category as chart signature
  - e.g., $ADJ : \lambda x. \text{fun}(x)$

• In practice, prune to top-N for each span
  - Approximate, but polynomial time
More on CCGs

- Generalized type-raising operations
- Cross composition operations for cross serial dependencies
- Compositional approaches to English intonation
- and a lot more ... even Jazz

[Steedman 1996; 2000; 2011; Granroth and Steedman 2012]
• Lambda calculus
• Parsing with Combinatory Categorial Grammars
• Linear CCGs
• Factored lexicons
Learning

- What kind of data/supervision can we use?
- What do we need to learn?
### Parsing as Structure Prediction

<table>
<thead>
<tr>
<th>show me flights to Boston</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S/N$</td>
</tr>
<tr>
<td>$\lambda f.f$</td>
</tr>
<tr>
<td>$N$</td>
</tr>
<tr>
<td>$\lambda x.\text{flight}(x)$</td>
</tr>
<tr>
<td>$PP/NP$</td>
</tr>
<tr>
<td>$\lambda y.\lambda x.\text{to}(x, y)$</td>
</tr>
<tr>
<td>$NP$</td>
</tr>
<tr>
<td>$\text{BOSTON}$</td>
</tr>
</tbody>
</table>

1. $\frac{PP}{\lambda x.\text{to}(x, \text{BOSTON})}$
2. $\frac{\lambda f.\lambda x.f(x) \land \text{to}(x, \text{BOSTON})}{N\!\setminus N}$
3. $\frac{\lambda x.\text{flight}(x) \land \text{to}(x, \text{BOSTON})}{N}$
4. $\frac{\lambda x.\text{flight}(x) \land \text{to}(x, \text{BOSTON})}{S}$
Learning CCG

\[
\begin{array}{cccc}
\text{show me} & \text{flights} & \text{to} & \text{Boston} \\
S/N & N & PP/NP & NP \\
\lambda f. f & \lambda x. \text{flight}(x) & \lambda y. \lambda x. \text{to}(x, y) & \text{BOSTON} \\
& & \lambda x. \text{to}(x, \text{BOSTON}) & \rightarrow \\
& & \lambda f. \lambda x. f(x) \land \text{to}(x, \text{BOSTON}) & \rightarrow \\
& & \lambda x. \text{flight}(x) \land \text{to}(x, \text{BOSTON}) & \rightarrow \\
& & \lambda x. \text{flight}(x) \land \text{to}(x, \text{BOSTON}) & \rightarrow \\
\end{array}
\]
Supervised Data

\[
\begin{array}{c|c|c|c}
\text{show me} & \text{flights} & \text{to} & \text{Boston} \\
S/N & N & PP/NP & NP \\
\lambda f. f & \lambda x. flight(x) & \lambda y. \lambda x. to(x, y) & BOSTON \\
& \lambda x. to(x, BOSTON) & \lambda f. \lambda x.f(x) \land to(x, BOSTON) & \lambda x. flight(x) \land to(x, BOSTON) \\
& \lambda x. flight(x) \land to(x, BOSTON) & \lambda x. flight(x) \land to(x, BOSTON) & \lambda x. flight(x) \land to(x, BOSTON) \\
\end{array}
\]
Supervised Data

\[
\begin{array}{llll}
\text{show me} & \text{flights} & \text{to} & \text{Boston} \\
S/N & N & PP/NP & NP \\
\lambda f.f & \lambda x.\text{flight}(x) & \lambda y.\lambda x.\text{to}(x, y) & \lambda x.\text{to}(x, \text{BOSTON}) \\
\end{array}
\]
Supervised Data

Supervised learning is done from pairs of sentences and logical forms

Show me flights to Boston
\[ \lambda x. \text{flight}(x) \land \text{to}(x, \text{BOSTON}) \]

I need a flight from baltimore to seattle
\[ \lambda x. \text{flight}(x) \land \text{from}(x, \text{BALTIMORE}) \land \text{to}(x, \text{SEATTLE}) \]

what ground transportation is available in san francisco
\[ \lambda x. \text{ground.transport}(x) \land \text{to.city}(x, \text{SF'}) \]

[Zettlemoyer and Collins 2005; 2007]
Weak Supervision

- Logical form is latent
- “Labeling” requires less expertise
- Labels don’t uniquely determine correct logical forms
- Learning requires executing logical forms within a system and evaluating the result
Weak Supervision
Learning from Query Answers

What is the largest state that borders Texas?

New Mexico

[Clarke et al. 2010; Liang et al. 2011]
Weak Supervision
Learning from Query Answers

What is the largest state that borders Texas?

New Mexico

\[
\text{argmax}(\lambda x.\text{state}(x)\,\land\, \text{border}(x, TX), \lambda y.\text{size}(y))
\]

\[
\text{argmax}(\lambda x.\text{river}(x)\,\land\, \text{in}(x, TX), \lambda y.\text{size}(y))
\]

[Clarke et al. 2010; Liang et al. 2011]
What is the largest state that borders Texas?

*New Mexico*

\[
\text{argmax} \left( \lambda x. \text{state}(x) \right) \\
\wedge \text{border}(x, TX), \lambda y. \text{size}(y))
\]

\[
\text{argmax} \left( \lambda x. \text{river}(x) \right) \\
\wedge \text{in}(x, TX), \lambda y. \text{size}(y))
\]

[Clarke et al. 2010; Liang et al. 2011]
Weak Supervision
Learning from Query Answers

What is the largest state that borders Texas?

New Mexico

\[ \text{argmax} (\lambda x. \text{state}(x) \land \text{border}(x, TX), \lambda y. \text{size}(y)) \]

\[ \text{argmax} (\lambda x. \text{river}(x) \land \text{in}(x, TX), \lambda y. \text{size}(y)) \]

[Clarke et al. 2010; Liang et al. 2011]
Weak Supervision
Learning from Demonstrations

at the chair, move forward three steps past the sofa

[Chen and Mooney 2011; Kim and Mooney 2012; Artzi and Zettlemoyer 2013b]
Weak Supervision
Learning from Demonstrations

at the chair, move forward three steps past the sofa

Some examples from other domains:

• Sentences and labeled game states [Goldwasser and Roth 2011]
• Sentences and sets of physical objects [Matuszek et al. 2012]

[Chen and Mooney 2011; Kim and Mooney 2012; Artzi and Zettlemoyer 2013b]
Weak Supervision
Learning from Conversation Logs

**System**: how can I help you? *(OPEN_TASK)*

**User**: i’d like to fly to new york

**System**: flying to new york. *(CONFIRM: from(fl, ATL))* leaving what city? *(ASK: λx. from(fl, x))*

**User**: from boston on june seven with american airlines

**System**: flying to new york. *(CONFIRM: to(fl, NYC))* what date would you like to depart boston? *(ASK: λx. date(fl, x) ∧ to(fl, BOS))*

**User**: june seventh

[CONVERSATION CONTINUES]
- Structured perceptron
- A unified learning algorithm
- Supervised learning
- Weak supervision Online
- Online
Structured Perceptron

- Simple additive updates
  - Only requires efficient decoding ($\text{argmax}$)
  - Closely related to MaxEnt and other feature rich models
  - Provably finds linear separator in finite updates, if one exists
- Challenge: learning with hidden variables
Structured Perceptron

- Simple additive updates
  - Only requires efficient decoding (argmax)
  - Closely related to MaxEnt and other feature rich models
  - Provably finds linear separator in finite updates, if one exists

- Challenge: learning with hidden variables

Derivations in the complete tutorial
Hidden Variable Perceptron

• No known convergence guarantees
  - Log-linear version is non-convex

• Simple and easy to implement
  - Works well with careful initialization

• Modifications for semantic parsing
  - Lots of different hidden information
  - Can add a margin constraint, do probabilistic version, etc.
Unified Learning Algorithm

• Handle various learning signals
• Estimate parsing parameters
• Induce lexicon structure
• Related to loss-sensitive structured perceptron [Singh-Miller and Collins 2007]
Learning Choices

**Validation Function**

\[ \mathcal{V} : \mathcal{Y} \rightarrow \{ t, f \} \]

- Indicates correctness of a parse \( y \)
- Varying \( \mathcal{V} \) allows for differing forms of supervision

**Lexical Generation Procedure**

\[ GENLEX (x, \mathcal{V}; \Lambda, \theta) \]

- Given:
  - sentence \( x \)
  - validation function \( \mathcal{V} \)
  - lexicon \( \Lambda \)
  - parameters \( \theta \)
- Produce a overly general set of lexical entries
Unified Learning Algorithm

Initialize $\theta$ using $\Lambda_0$, $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

- **Step 1:** (Lexical generation)
- **Step 2:** (Update parameters)

**Output:** Parameters $\theta$ and lexicon $\Lambda$

- **Online**
- **2 steps:**
  - Lexical generation
  - Parameter update
Initialize $\theta$ using $\Lambda_0$, $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T$, $i = 1 \ldots n$:

- **Step 1:** (Lexical generation)
- **Step 2:** (Update parameters)

**Output:** Parameters $\theta$ and lexicon $\Lambda$
Initialize $\theta$ using $\Lambda_0$, $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T$, $i = 1 \ldots n$:

- **Step 1**: (Lexical generation)
- **Step 2**: (Update parameters)

**Output**: Parameters $\theta$ and lexicon $\Lambda$
Initialize $\theta$ using $\Lambda_0$, $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

**Step 1:** (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, V_i; \Lambda, \theta)$,
   $\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let $Y$ be the $k$ highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
   $\lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y;\theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

**Step 2:** (Update parameters)

**Output:** Parameters $\theta$ and lexicon $\Lambda$
Initialize $\theta$ using $\Lambda_0$, $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T$, $i = 1 \ldots n$:

**Step 1:** (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, V_i; \Lambda, \theta)$, $\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let $Y$ be the $k$ highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
   $$\lambda_i \leftarrow \bigcup_{y \in \text{MAXV}_i(Y; \theta)} LEX(y)$$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

**Step 2:** (Update parameters)

Output: Parameters $\theta$ and lexicon $\Lambda$

---

Generate a large set of potential lexical entries

- $\theta$ weights
- $x$ sentence
- $V$ validation function
- $GENLEX(x, V; \lambda, \theta)$
  
  lexical generation function
Initialize $\theta$ using $\Lambda_0$ , $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T$, $i = 1 \ldots n$ :

**Step 1:** (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, V_i; \Lambda, \theta)$,
   $\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let $Y$ be the $k$ highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
   $\lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y; \theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

**Step 2:** (Update parameters)

Output: Parameters $\theta$ and lexicon $\Lambda$

Generate a large set of potential lexical entries

\[ \theta \] weights
\[ x \] sentence
\[ \mathcal{V} \] validation function
\[ GENLEX(x, \mathcal{V}; \lambda, \theta) \]
lexical generation function

\[ \mathcal{V} : \mathcal{V} \rightarrow \{ t, f \} \]
\[ \mathcal{V} \] all parses
Initialize $\theta$ using $\Lambda_0$, $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T$, $i = 1 \ldots n$:

**Step 1:** (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, V_i; \Lambda, \theta)$,
   $\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let $Y$ be the $k$ highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
   $\lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y; \theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

**Step 2:** (Update parameters)

Output: Parameters $\theta$ and lexicon $\Lambda$

Generate a large set of potential lexical entries

- $\theta$ weights
- $x$ sentence
- $V$ validation function
- $GENLEX(x, V; \lambda, \theta)$
  lexical generation function

Procedure to propose potential new lexical entries for a sentence
Initialize $\theta$ using $\Lambda_0$, $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T$, $i = 1 \ldots n$:

**Step 1:** (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$, $\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let $Y$ be the $k$ highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
   $\lambda_i \leftarrow \bigcup_{y \in \text{MAXV}_i(Y; \theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

**Step 2:** (Update parameters)

Output: Parameters $\theta$ and lexicon $\Lambda$
Initialize $\theta$ using $\Lambda_0$, $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

**Step 1:** (Lexical generation)

a. Set $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$,
   $\lambda \leftarrow \Lambda \cup \lambda_G$

b. Let $Y$ be the $k$ highest scoring parses from $GEN(x_i; \lambda)$

c. Select lexical entries from the highest scoring valid parses:
   $\lambda_i \leftarrow \bigcup_{y \in \text{MAX}_i(Y;\theta)} LEX(y)$

d. Update lexicon: $\Lambda \leftarrow \Lambda \cup \lambda_i$

**Step 2:** (Update parameters)

**Output:** Parameters $\theta$ and lexicon $\Lambda$

Get lexical entries from highest scoring valid parses
Initialize \( \theta \) using \( \Lambda_0 \), \( \Lambda \leftarrow \Lambda_0 \)

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

**Step 1:** (Lexical generation)

a. Set \( \lambda_G \leftarrow GENLEX(x_i, V_i; \Lambda, \theta) \),
   \( \lambda \leftarrow \Lambda \cup \lambda_G \)

b. Let \( Y \) be the \( k \) highest scoring parses from \( GEN(x_i; \lambda) \)

c. Select lexical entries from the highest scoring valid parses:
   \( \lambda_i \leftarrow \bigcup_{y \in MAXV_i(Y; \theta)} LEX(y) \)

d. Update lexicon: \( \Lambda \leftarrow \Lambda \cup \lambda_i \)

**Step 2:** (Update parameters)

**Output:** Parameters \( \theta \) and lexicon \( \Lambda \)
Initialize $\theta$ using $\Lambda_0$, $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

**Step 1:** (Lexical generation)

**Step 2:** (Update parameters)

a. Set $G_i \leftarrow MAXV_i(GEN(x_i; \Lambda); \theta)$
   and $B_i \leftarrow \{ e | e \in GEN(x_i; \Lambda) \land \neg \nu_i(y) \}$

b. Construct sets of margin violating good and bad parses:
   
   - $R_i \leftarrow \{ g | g \in G_i \land \exists b \in B_i$
   
   - $s.t. \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b) \}$

   - $E_i \leftarrow \{ b | b \in B_i \land \exists g \in G_i$
   
   - $s.t. \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b) \}$

c. Apply the additive update:
   
   $$\theta \leftarrow \theta + \frac{1}{|R_i|} \sum_{r \in R_i} \Phi_i(r)$$

   $$- \frac{1}{|E_i|} \sum_{e \in E_i} \Phi_i(e)$$

**Output:** Parameters $\theta$ and lexicon $\Lambda$
Initialize $\theta$ using $\Lambda_0$, $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T$, $i = 1 \ldots n$:

**Step 1:** (Lexical generation)

**Step 2:** (Update parameters)

a. Set $G_i \leftarrow MAXV_i(GEN(x_i; \Lambda); \theta)$
   and $B_i \leftarrow \{e | e \in GEN(x_i; \Lambda) \land \neg V_i(y)\}$

b. Construct sets of margin violating good and bad parses:
   
   $R_i \leftarrow \{g | g \in G_i \land \exists b \in B_i$
   
   $\text{s.t. } \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$

   $E_i \leftarrow \{b | b \in B_i \land \exists g \in G_i$
   
   $\text{s.t. } \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$

c. Apply the additive update:
   
   $\theta \leftarrow \theta + \frac{1}{|R_i|} \sum_{r \in R_i} \Phi_i(r)$

   $- \frac{1}{|E_i|} \sum_{e \in E_i} \Phi_i(e)$

**Output:** Parameters $\theta$ and lexicon $\Lambda$

---

Re-parse and group all parses into ‘good’ and ‘bad’ sets

$\theta$ weights

$x$ sentence

$V$ validation function

$GEN(x; \lambda)$ set of all parses

$MAXV_i(Y; \theta) =$

$\{y | \forall y' \in Y, \langle \theta, \Phi_i(y') \rangle \leq \langle \theta, \Phi_i(y) \rangle \land$

$V_i(y) = 1\}$
Initialize $\theta$ using $\Lambda_0$, $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

**Step 1:** (Lexical generation)

**Step 2:** (Update parameters)

a. Set $G_i \leftarrow MAXV_i(\text{GEN}(x_i; \Lambda); \theta)$
   
   and $B_i \leftarrow \{e|e \in \text{GEN}(x_i; \Lambda) \land \lnot V_i(y)\}$

b. Construct sets of margin violating good and bad parses:

   $R_i \leftarrow \{g|g \in G_i \land \exists b \in B_i$
   
   $\text{s.t. } \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$

   $E_i \leftarrow \{b|b \in B_i \land \exists g \in G_i$
   
   $\text{s.t. } \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$

   c. Apply the additive update:

   $\theta \leftarrow \theta + \frac{1}{|R_i|} \sum_{r \in R_i} \Phi_i(r) - \frac{1}{|E_i|} \sum_{e \in E_i} \Phi_i(e)$

**Output:** Parameters $\theta$ and lexicon $\Lambda$

For all pairs of ‘good’ and ‘bad’ parses, if their scores violate the margin, add each to ‘right’ and ‘error’ sets respectively

$\theta$ weights

$\gamma$ margin

$\phi_i(y) = \phi(x_i, y)$

$\Delta_i(y, y') = |\Phi_i(y) - \Phi_i(y')|_1$
Initialize $\theta$ using $\Lambda_0$, $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$:

**Step 1:** (Lexical generation)

**Step 2:** (Update parameters)

a. Set $G_i \leftarrow \text{MAXV}_i(\text{GEN}(x_i; \Lambda); \theta)$
   and $B_i \leftarrow \{ e | e \in \text{GEN}(x_i; \Lambda) \land \neg \mathcal{V}_i(y) \}$

b. Construct sets of margin violating good and bad parses:
   
   $R_i \leftarrow \{ g | g \in G_i \land \exists b \in B_i$ 
   s.t. $\langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b) \}$

   $E_i \leftarrow \{ b | b \in B_i \land \exists g \in G_i$ 
   s.t. $\langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b) \}$

   c. Apply the additive update:

   $\theta \leftarrow \theta + \frac{1}{|R_i|} \sum_{r \in R_i} \Phi_i(r)$
   $- \frac{1}{|E_i|} \sum_{e \in E_i} \Phi_i(e)$

**Output:** Parameters $\theta$ and lexicon $\Lambda$

---

**Update towards violating ‘good’ parses and against violating ‘bad’ parses**

$\theta$ weights

$\phi_i(y) = \phi(x_i, y)$
Initialize $\theta$ using $\Lambda_0$, $\Lambda \leftarrow \Lambda_0$

For $t = 1 \ldots T, i = 1 \ldots n$ :

**Step 1:** (Lexical generation)

**Step 2:** (Update parameters)

**Output:** Parameters $\theta$ and lexicon $\Lambda$

Return grammar

$\theta$ weights

$\Lambda$ lexicon
Features and Initialization

**Feature Classes**
- Parse: indicate lexical entry and combinator use
- Logical form: indicate local properties of logical forms, such as constant co-occurrence

**Lexicon Initialization**
- Often use an NP list
- Sometimes include additional, domain independent entries for function words

**Initial Weights**
- Positive weight for initial lexical indicator features
Unified Learning Algorithm

Extensions

• Loss-sensitive learning
  - Applied to learning from conversations

• Stochastic gradient descent
  - Approximate expectation computation

[Artzi and Zettlemoyer 2011; Zettlemoyer and Collins 2005]
Unified Learning Algorithm

\[ \mathcal{V} \text{ validation function} \]

\[ GENLEX(x, \mathcal{V}; \lambda, \theta) \]

lexical generation function

- Two parts of the algorithm we still need to define
- Depend on the task and supervision signal
## Unified Learning Algorithm

### Supervised

- Template-based *GENLEX*
- Unification-based *GENLEX*

### Weakly Supervised

- Template-based *GENLEX*
Supervised Learning

show me the afternoon flights from LA to boston

\( \lambda x. \text{flight}(x) \land \text{during}(x, \text{AFTERNOON}) \land \text{from}(x, \text{LA}) \land \text{to}(x, \text{BOS}) \)
Supervised Learning

show me the afternoon flights from LA to boston

\( \lambda x. \text{flight}(x) \land \text{during}(x, \text{AFTERNOON}) \land \text{from}(x, \text{LA}) \land \text{to}(x, \text{BOS}) \)

Parse structure is latent
Supervised Validation Function

- Validate logical form against gold label

\[ \mathcal{V}_i(y) = \begin{cases} 
  \text{true} & \text{if } LF(y) = z_i \\
  \text{false} & \text{else}
\end{cases} \]

- \( y \) parse
- \( z_i \) labeled logical form
- \( LF(y) \) logical form at the root of \( y \)
Supervised Template-based

\[ \text{GENLEX}(x, z; \Lambda, \theta) \]

- Sentence
- Logical form
- Lexicon
- Weights

Small notation abuse: take labeled logical form instead of validation function
Supervised Template-based

\( \text{GENLEX}(x, z; \Lambda, \theta) \)

I want a flight to new york

\( \lambda x. \text{flight}(x) \land \text{to}(x, \text{NYC}) \)
Supervised Template-based GENLEX

• Use templates to constrain lexical entries structure

• For example: from a small annotated dataset

\[
\begin{align*}
\lambda(\omega, \{v_i\}_1^n).[\omega \vdash ADJ : \lambda x.v_1(x)] \\
\lambda(\omega, \{v_i\}_1^n).[\omega \vdash PP : \lambda x.\lambda y.v_1(y, x)] \\
\lambda(\omega, \{v_i\}_1^n).[\omega \vdash N : \lambda x.v_1(x)] \\
\lambda(\omega, \{v_i\}_1^n).[\omega \vdash S\backslash NP/NP : \lambda x.\lambda y.v_1(x, y)]
\end{align*}
\] ...
Supervised Template-based GENLEX

Need lexemes to instantiate templates

\[
\lambda(\omega, \{v_i\}_1^n).[\omega \vdash ADJ : \lambda x. v_1(x)]
\]

\[
\lambda(\omega, \{v_i\}_1^n).[\omega \vdash PP : \lambda x. \lambda y. v_1(y, x)]
\]

\[
\lambda(\omega, \{v_i\}_1^n).[\omega \vdash N : \lambda x. v_1(x)]
\]

\[
\lambda(\omega, \{v_i\}_1^n).[\omega \vdash S\backslash NP/NP : \lambda x. \lambda y. v_1(x, y)]
\]

...
Supervised Template-based

\[ \text{GENLEX}(x, z; \Lambda, \theta) \]

\[ \lambda x. \text{flight}(x) \land \text{to}(x, \text{NYC}) \]

I want a flight to new york

I want
a flight
flight
flight to new
...

All possible sub-strings
Supervised Template-based

\[ GENLEX(x, z; \Lambda, \theta) \]

I want a flight to new york

\[ \lambda x. \text{flight}(x) \land \text{to}(x, \text{NYC}) \]

I want a flight
flight
flight to new
NYC
Supervised Template-based

\[ \text{GENLEX}(x, z; \Lambda, \theta) \]

I want a flight to new york

\[ \lambda x. \text{flight}(x) \land \text{to}(x, \text{NYC}) \]

I want

a flight

flight

flight to new

\ldots

Create lexemes

\( (\text{flight}, \{\text{flight}\}) \)

(I want, {})

(\text{flight to new}, \{\text{to, NYC}\})

\ldots
Supervised Template-based

\( GENLEX(x, z; \Lambda, \theta) \)

I want a flight to new york

\( \lambda x. \text{flight}(x) \land \text{to}(x, \text{NYC}) \)

I want

flight

flight to new

... (flight, \{flight\})

(I want, \{\})

(flight to new, \{to, NYC\})

... ...

Initialize templates

flight \triangleright N : \lambda x. \text{flight}(x)

I want \triangleright S/NP : \lambda x.x

flight to new : S/NP/NP : \lambda x.\lambda y.\text{to}(x, y)

...
Fast Parsing with Pruning

- GENLEX outputs a large number of entries
- For fast parsing: use the labeled logical form to prune
- Prune partial logical forms that can’t lead to labeled form

I want a flight from New York to Boston on Delta

\[ \lambda x. \text{from}(x, NYC) \land \text{to}(x, BOS) \land \text{carrier}(x, DL) \]
Fast Parsing with Pruning

I want a flight from New York to Boston on Delta

$$\lambda x. \text{from}(x, NYC) \land \text{to}(x, BOS) \land \text{carrier}(x, DL)$$

<table>
<thead>
<tr>
<th>Form</th>
<th>New York</th>
<th>To</th>
<th>Boston</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP/NP</td>
<td>NYC</td>
<td>PP/NP</td>
<td>BOS</td>
</tr>
<tr>
<td>$$\lambda x. \lambda y. \text{to}(y, x)$$</td>
<td>$$\lambda x. \lambda y. \text{to}(y, x)$$</td>
<td>$$\lambda x. \lambda y. \text{to}(y, x)$$</td>
<td></td>
</tr>
</tbody>
</table>
I want a flight from New York to Boston on Delta

\( \lambda x. \text{from}(x, \text{NYC}) \land \text{to}(x, \text{BOS}) \land \text{carrier}(x, \text{DL}) \)

\[
\begin{array}{cccccc}
\ldots & \text{form} & \text{New York} & \text{to} & \text{Boston} & \ldots \\
& PP/NP & NP & PP/NP & NP & \\
& \lambda x. \lambda y. \text{to}(y, x) & \text{NYC} & \lambda x. \lambda y. \text{to}(y, x) & \text{BOS} & \\
& PP & \Rightarrow & PP & \Rightarrow \\
& \lambda y. \text{to}(y, \text{NYC}) & & \lambda y. \text{to}(y, \text{BOS}) & & \\
\end{array}
\]
Fast Parsing with Pruning

I want a flight from New York to Boston on Delta

\( \lambda x. \text{from}(x, \text{NYC}) \land \text{to}(x, \text{BOS}) \land \text{carrier}(x, \text{DL}) \)

\[ \begin{array}{c}
\text{form} \\
PP/\text{NP}
\end{array} \quad \begin{array}{c}
\text{New York} \\
\text{NP} \\
\text{NYC}
\end{array} \quad \begin{array}{c}
\text{to} \\
PP/\text{NP}
\end{array} \quad \begin{array}{c}
\text{Boston} \\
\text{NP} \\
\text{BOS}
\end{array} \quad \ldots
\]

\[ \begin{array}{c}
\lambda x. \lambda y. \text{to}(y, x) \\
\lambda y. \text{to}(y, \text{NYC})
\end{array} \quad \begin{array}{c}
\text{PP} \\
\lambda y. \text{to}(y, \text{NYC})
\end{array} \quad \begin{array}{c}
\text{PP} \\
\lambda y. \text{to}(y, \text{BOS})
\end{array} \quad \ldots
\]
Fast Parsing with Pruning

I want a flight from New York to Boston on Delta

$$\lambda x. \text{from}(x, \text{NYC}) \land \text{to}(x, \text{BOS}) \land \text{carrier}(x, \text{DL})$$

... form New York to Boston ...

\[
\begin{array}{c}
\text{PP/NN} \\
\lambda x. \lambda y. \text{to}(y, x)
\end{array}
\]

\[
\begin{array}{c}
\text{NP} \\
\text{NYC}
\end{array}
\]

\[
\begin{array}{c}
\times \\
\times
\end{array}
\]

\[
\begin{array}{c}
\text{PP} \\
\lambda y. \text{to}(y, \text{NYC})
\end{array}
\]

\[
\begin{array}{c}
\text{PP/NN} \\
\lambda x. \lambda y. \text{to}(y, x)
\end{array}
\]

\[
\begin{array}{c}
\text{NP} \\
\text{BOS}
\end{array}
\]

\[
\begin{array}{c}
\text{PP} \\
\lambda y. \text{to}(y, \text{BOS})
\end{array}
\]

\[
\begin{array}{c}
\text{N\backslash N} \\
\lambda f. \lambda y. f(y) \land \text{to}(y, \text{BOS})
\end{array}
\]
Supervised Template-based GENLEX

Summary

<table>
<thead>
<tr>
<th>Feature</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No initial expert knowledge</td>
<td>✓</td>
</tr>
<tr>
<td>Creates compact lexicons</td>
<td>✓</td>
</tr>
<tr>
<td>Language independent</td>
<td></td>
</tr>
<tr>
<td>Representation independent</td>
<td></td>
</tr>
<tr>
<td>Easily inject linguistic knowledge</td>
<td>✓</td>
</tr>
<tr>
<td>Weakly supervised learning</td>
<td>✓</td>
</tr>
</tbody>
</table>
Unification-based GENLEX

• Automatically learns the templates
  - Can be applied to any language and many different approaches for semantic modeling

• Two step process
  - Initialize lexicon with labeled logical forms
  - “Reverse” parsing operations to split lexical entries

[Kwiatkowski et al. 2010]
Unification-based GENLEX

- Initialize lexicon with labeled logical forms

For every labeled training example:

I want a flight to Boston

\[ \lambda x. \text{flight}(x) \land \text{to}(x, BOS) \]

Initialize the lexicon with:

I want a flight to Boston \[\vdash S : \lambda x. \text{flight}(x) \land \text{to}(x, BOS)\]
Unification-based GENLEX

- Splitting lexical entries

I want a flight to Boston ⊨ S : λx.\textit{flight}(x) \land \textit{to}(x, BOS)

I want a flight ⊨ S/(S\mid NP) : λf.λx.\textit{flight}(x) \land f(x)

to Boston ⊨ S\mid NP : λx.\textit{to}(x, BOS)
Unification-based GENLEX

• Splitting lexical entries

I want a flight to Boston \vdash S : \lambda x. \text{flight}(x) \land\text{to}(x, BOS)

I want a flight \vdash S/(S|NP) : \lambda f. \lambda x. \text{flight}(x) \land f(x)

to Boston \vdash S|NP : \lambda x. \text{to}(x, BOS)

Many possible phrase pairs \times Many possible category pairs
Unification-based GENLEX

• Splitting lexical entries

I want a flight to Boston ⊨ S : \( \lambda x. \text{flight}(x) \land \text{to}(x, \text{BOS}) \)

I want a flight ⊨ \( S/(S|NP) : \lambda f. \lambda x.f(x) \)

to Boston ⊨ \( S|NP : \lambda x. \text{to}(x, \text{BOS}) \)

Many possible phrase pairs

More details in the complete tutorial
Experiments

• Two database corpora:
  - Geo880/Geo250 [Zelle and Mooney 1996; Tang and Mooney 2001]
  - ATIS [Dahl et al. 1994]

• Learning from sentences paired with logical forms

• Comparing template-based and unification-based GENLEX methods

[Zettlemoyer and Collins 2007; Kwiatkowski et al. 2010; 2011]
Results

Template-based | Unification-based | Unification-based + Factored Lexicon

Geo880 | ATIS | Geo250 English | Geo250 Spanish | Geo250 Japanese | Geo250 Turkish

[Zettlemoyer and Collins 2007; Kwiatkowski et al. 2010; 2011]
# GENLEX Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Templates</th>
<th>Unification</th>
</tr>
</thead>
<tbody>
<tr>
<td>No initial expert knowledge</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Creates compact lexicons</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Language independent</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Representation independent</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Easily inject linguistic knowledge</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Weakly supervised learning</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
## GENLEX Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Templates</th>
<th>Unification</th>
</tr>
</thead>
<tbody>
<tr>
<td>No initial expert knowledge</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Creates compact lexicons</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Language independent</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Representation independent</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Easily inject linguistic knowledge</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Weakly supervised learning</td>
<td>✓</td>
<td>?</td>
</tr>
</tbody>
</table>
• Structured perceptron
• A unified learning algorithm
• Supervised learning

- Weak supervision
  Online
Modeling

Show me all papers about semantic parsing

$\lambda x. \text{paper}(x) \land \text{topic}(x, SEMPAR)$
Modeling

Show me all papers about semantic parsing

 Parsing with CCG

\[ \lambda x. \text{paper}(x) \land \text{topic}(x, SEMPAR) \]

What should these logical forms look like?

But why should we care?
Modeling Considerations

Modeling is key to learning compact lexicons and high performing models

• Capture language complexity
• Satisfy system requirements
• Align with language units of meaning
• Semantic modeling for:
  - Querying databases
  - Referring to physical objects
  - Executing instructions
Querying Databases

### State

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Capital</th>
<th>Pop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
<td>3.9</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
<td>0.4</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
<td>2.7</td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
<td>4.1</td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
<td>17.5</td>
</tr>
</tbody>
</table>

### Border

<table>
<thead>
<tr>
<th>State1</th>
<th>State2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>OR</td>
</tr>
<tr>
<td>WA</td>
<td>ID</td>
</tr>
<tr>
<td>CA</td>
<td>OR</td>
</tr>
<tr>
<td>CA</td>
<td>NV</td>
</tr>
</tbody>
</table>

### Mountains

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
</tr>
<tr>
<td>Wrangel</td>
<td>AK</td>
</tr>
<tr>
<td>Sill</td>
<td>CA</td>
</tr>
<tr>
<td>Elbert</td>
<td>CO</td>
</tr>
</tbody>
</table>

[Zettlemoyer and Collins 2005]
### Querying Databases

#### State

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Capital</th>
<th>Pop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
<td>3.9</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
<td>0.4</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
<td>2.7</td>
</tr>
</tbody>
</table>

#### Border

<table>
<thead>
<tr>
<th>State1</th>
<th>State2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>OR</td>
</tr>
<tr>
<td>WA</td>
<td>ID</td>
</tr>
<tr>
<td>CA</td>
<td>OR</td>
</tr>
<tr>
<td>CA</td>
<td>NV</td>
</tr>
</tbody>
</table>

#### Mountains

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
</tr>
</tbody>
</table>

What is the capital of Arizona?
How many states border California?
What is the largest state?
What is the capital of Arizona?

How many states border California?

What is the largest state?

Noun Phrases
What is the capital of Arizona?

How many states border California?

What is the largest state?
### Querying Databases

#### State

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Capital</th>
<th>Pop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
<td>3.9</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
<td>0.4</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
<td>2.7</td>
</tr>
</tbody>
</table>

#### Border

<table>
<thead>
<tr>
<th>State1</th>
<th>State2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>OR</td>
</tr>
<tr>
<td>WA</td>
<td>ID</td>
</tr>
<tr>
<td>CA</td>
<td>OR</td>
</tr>
<tr>
<td>CA</td>
<td>NV</td>
</tr>
</tbody>
</table>

#### Mountains

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
</tr>
</tbody>
</table>

---

**What is the capital of Arizona?**

**How many states border California?**

**What is the largest state?**
**Mountains**

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
</tr>
</tbody>
</table>

**State Abbr.-Capital-Pop.**

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Capital</th>
<th>Pop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
<td>3.9</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
<td>0.4</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
<td>2.7</td>
</tr>
</tbody>
</table>

**Border**

<table>
<thead>
<tr>
<th>State1</th>
<th>State2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>OR</td>
</tr>
<tr>
<td>WA</td>
<td>ID</td>
</tr>
<tr>
<td>CA</td>
<td>OR</td>
</tr>
<tr>
<td>CA</td>
<td>NV</td>
</tr>
</tbody>
</table>

**Querying Databases**

- What is the capital of Arizona?
- How many states border California?
- What is the largest state?

**Prepositions**
## Querying Databases

### State

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Capital</th>
<th>Pop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
<td>3.9</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
<td>0.4</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
<td>2.7</td>
</tr>
</tbody>
</table>

### Border

<table>
<thead>
<tr>
<th>State1</th>
<th>State2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>OR</td>
</tr>
<tr>
<td>WA</td>
<td>ID</td>
</tr>
<tr>
<td>CA</td>
<td>OR</td>
</tr>
<tr>
<td>CA</td>
<td>NV</td>
</tr>
</tbody>
</table>

### Mountains

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
</tr>
</tbody>
</table>

### Superlatives

What is the capital of Arizona?
How many states border California?
What is the largest state?
What is the capital of Arizona?
How many states border California?
What is the largest state?
# Querying Databases

<table>
<thead>
<tr>
<th>State</th>
<th>Abbr.</th>
<th>Capital</th>
<th>Pop.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AL</td>
<td>Montgomery</td>
<td>3.9</td>
</tr>
<tr>
<td></td>
<td>AK</td>
<td>Juneau</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>AZ</td>
<td>Phoenix</td>
<td>2.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State1</th>
<th>State2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>OR</td>
</tr>
<tr>
<td>WA</td>
<td>ID</td>
</tr>
<tr>
<td>CA</td>
<td>OR</td>
</tr>
<tr>
<td>CA</td>
<td>NV</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mountains</th>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td></td>
<td>Antero</td>
<td>CO</td>
</tr>
<tr>
<td></td>
<td>Rainier</td>
<td>WA</td>
</tr>
<tr>
<td></td>
<td>Shasta</td>
<td>CA</td>
</tr>
</tbody>
</table>

**Questions**

- What is the capital of Arizona?
- How many states border California?
- What is the largest state?
Referring to DB Entities

- Noun phrases: Select single DB entities
- Prepositions and Verbs: Relations between entities
- Nouns: Typing (i.e., column headers)
- Superlatives: Ordering queries
# Noun Phrases

## State

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
</tr>
</tbody>
</table>

## Mountains

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
</tr>
</tbody>
</table>

- **Noun phrases name specific entities**
  - Washington (WA)
  - Florida (FL)
  - The Sunshine State (FL)
Noun Phrases

State | Abbr. | Capital
---|---|---
AL | Montgomery
AK | Juneau
AZ | Phoenix
WA | Olympia
NY | Albany
IL | Springfield

Mountains

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
</tr>
</tbody>
</table>

Noun phrases name specific entities

Washington
WA

Florida
The Sunshine State
FL

e-typed entities
### Noun Phrases

#### State

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
</tr>
</tbody>
</table>

#### Mountains

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
</tr>
</tbody>
</table>

Noun phrases name specific entities

**Washington**

\[
NP_{WA}
\]

**The Sunshine State**

\[
NP_{FL}
\]
### Verb Relations

#### State

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
</tr>
</tbody>
</table>

#### Border

The table shows the border relations between states:

<table>
<thead>
<tr>
<th>State1</th>
<th>State2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>OR</td>
</tr>
<tr>
<td>WA</td>
<td>ID</td>
</tr>
<tr>
<td>CA</td>
<td>OR</td>
</tr>
<tr>
<td>CA</td>
<td>NV</td>
</tr>
</tbody>
</table>

*Verbs express relations between entities*

- **Nevada borders California**
  
  \[\text{border}(NV, CA)\]
Verb Relations

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Border</th>
</tr>
</thead>
<tbody>
<tr>
<td>State1</td>
</tr>
<tr>
<td>WA</td>
</tr>
<tr>
<td>WA</td>
</tr>
<tr>
<td>CA</td>
</tr>
<tr>
<td>CA</td>
</tr>
</tbody>
</table>

Verbs express relations between entities

Nevada **borders** California
\[\text{border}(NV, CA)\]

true
Vert Relations

<table>
<thead>
<tr>
<th>State</th>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
<td></td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
<td></td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
<td></td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
<td></td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
<td></td>
</tr>
</tbody>
</table>

**Nevada**

\[
\begin{align*}
S \setminus NP/\ NP \\
\lambda x. \lambda y. \text{border}(y, x)
\end{align*}
\]

**California**

\[
\begin{align*}
S \setminus NP \\
\lambda y. \text{border}(y, CA)
\end{align*}
\]

\[
\text{border}(NV, CA)
\]
Nouns

Nouns are functions that define entity type

state

\( \lambda x. \text{state}(x) \)

mountain

\( \lambda x. \text{mountain}(x) \)
Nouns are functions that define entity type

\[ \lambda x. \text{state}(x) \]
\[
\{ \text{WA}, \text{AL}, \text{AK}, \ldots \}
\]

\[ \lambda x. \text{mountain}(x) \]
\[
\{ \text{BIANCA}, \text{ANTERO}, \ldots \}
\]

functions define sets

\[ e \rightarrow t \]
### State

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
</tr>
</tbody>
</table>

### Mountains

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
</tr>
</tbody>
</table>

**Nouns**

Nouns are functions that define entity type.

\[
\text{state} \quad \lambda x. \text{state}(x)
\]

\[
\text{mountain} \quad \lambda x. \text{mountain}(x)
\]
### Prepositions

Prepositional phrases are conjunctive modifiers

#### Mountain in Colorado

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
</tr>
</tbody>
</table>

#### State Capital Table

<table>
<thead>
<tr>
<th>State</th>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
<td></td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
<td></td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
<td></td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
<td></td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
<td></td>
</tr>
</tbody>
</table>
### Prepositions

#### Mountains

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
</tr>
</tbody>
</table>

Prepositional phrases are conjunctive modifiers

\[
\lambda x. \text{mountain}(x)
\]

\{ BIANCA, ANTERO, RAINIER, \ldots \}
## Prepositions

Prepositional phrases are conjunctive modifiers

\[
\lambda x. \text{mountain}(x) \land \text{in}(x, \text{CO})
\]

\{ \text{BIANCA}, \text{ANTERO} \}
## Prepositions

<table>
<thead>
<tr>
<th>State</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
</tr>
</tbody>
</table>

\[
\lambda x. \text{mountain}(x) \quad \text{in} \quad \lambda y. \lambda x. \text{in}(x, y) \quad \text{Colorado} \\
\]

\[
\text{mountain} \quad \frac{\lambda x. \text{mountain}(x) \wedge \text{in}(x, \text{CO})}{\lambda x. \text{mountain}(x) \wedge \text{in}(x, \text{CO})} \\
\frac{\lambda y. \lambda x. \text{in}(x, y)}{\text{in} \frac{\text{in}(x, \text{CO})}{\text{CO}}} \\
\]
## Function Words

### State

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
</tr>
</tbody>
</table>

### Border

<table>
<thead>
<tr>
<th>State I</th>
<th>State2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>OR</td>
</tr>
<tr>
<td>WA</td>
<td>ID</td>
</tr>
<tr>
<td>CA</td>
<td>OR</td>
</tr>
<tr>
<td>CA</td>
<td>NV</td>
</tr>
</tbody>
</table>

Certain words are used to modify syntactic roles:

\[
\lambda x. \text{state}(x) \land \text{border}(x, CA) \left\{ \text{OR, NV, AZ} \right\}
\]
Function Words
## Function Words

### State

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
</tr>
</tbody>
</table>

### Border

<table>
<thead>
<tr>
<th>State1</th>
<th>State2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>OR</td>
</tr>
<tr>
<td>WA</td>
<td>ID</td>
</tr>
<tr>
<td>CA</td>
<td>OR</td>
</tr>
<tr>
<td>CA</td>
<td>NV</td>
</tr>
</tbody>
</table>

Certain words are used to modify syntactic roles

- May have other senses with semantic meaning
- May carry content in other domains

Other common function words: which, of, for, are, is, does, please
Definite Determiners

<table>
<thead>
<tr>
<th>State</th>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>AL</td>
<td>Montgomery</td>
</tr>
<tr>
<td>AK</td>
<td>AK</td>
<td>Juneau</td>
</tr>
<tr>
<td>AZ</td>
<td>AZ</td>
<td>Phoenix</td>
</tr>
<tr>
<td>WA</td>
<td>WA</td>
<td>Olympia</td>
</tr>
<tr>
<td>NY</td>
<td>NY</td>
<td>Albany</td>
</tr>
<tr>
<td>IL</td>
<td>IL</td>
<td>Springfield</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mountains</th>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
<td></td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
<td></td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
<td></td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
<td></td>
</tr>
</tbody>
</table>

Definite determiner selects the single members of a set when such exists

\[ \iota: \ (e \rightarrow t) \rightarrow e \]

the mountain in Washington
Definite Determiners

Definite determiner selects the single members of a set when such exists

\[ \iota : (e \rightarrow t) \rightarrow e \]

mountain in Washington

\[ \lambda x. \text{mountain}(x) \land \text{in}(x, \text{WA}) \]

\{ \text{RAINIER} \}
Definite Determiners

Definite determiner selects the single members of a set when such exists

\[ \iota : (e \rightarrow t) \rightarrow e \]

the mountain in Washington

\[ \iota x. \text{mountain}(x) \land \text{in}(x, WA) \]

Definite determiner

State

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
</tr>
</tbody>
</table>

Mountains

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
</tr>
</tbody>
</table>
**Definite Determiners**

<table>
<thead>
<tr>
<th>State</th>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>AL</td>
<td>Montgomery</td>
</tr>
<tr>
<td>AK</td>
<td>AK</td>
<td>Juneau</td>
</tr>
<tr>
<td>AZ</td>
<td>AZ</td>
<td>Phoenix</td>
</tr>
<tr>
<td>WA</td>
<td>WA</td>
<td>Olympia</td>
</tr>
<tr>
<td>NY</td>
<td>NY</td>
<td>Albany</td>
</tr>
<tr>
<td>IL</td>
<td>IL</td>
<td>Springfield</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mountains</th>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
<td></td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
<td></td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
<td></td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
<td></td>
</tr>
</tbody>
</table>

Definite determiner selects the single members of a set when such exists:

\[ \lambda : (e \rightarrow t) \rightarrow e \]

The mountain in Colorado:

\[ \lambda x. \text{mountain}(x) \land \text{in}(x, CO) \]

\[ \{ \text{BIANCA}, \text{ANTERO} \} \rightarrow ? \]
Definite Determiners

**State**

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
</tr>
</tbody>
</table>

**Mountains**

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td>Antero</td>
<td>CO</td>
</tr>
<tr>
<td>Rainier</td>
<td>WA</td>
</tr>
<tr>
<td>Shasta</td>
<td>CA</td>
</tr>
</tbody>
</table>

Definite determiner selects the single members of a set when such exists:

\[ \iota : (e \rightarrow t) \rightarrow e \]

the mountain in Colorado:

\[ \iota x. \text{mountain}(x) \land \text{in}(x, \text{CO}) \]

\{ BIANCA, ANTERO \} \rightarrow \times

No information to disambiguate
## Definite Determiners

<table>
<thead>
<tr>
<th>State</th>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AL</td>
<td>Montgomery</td>
</tr>
<tr>
<td></td>
<td>AK</td>
<td>Juneau</td>
</tr>
<tr>
<td></td>
<td>AZ</td>
<td>Phoenix</td>
</tr>
<tr>
<td></td>
<td>WA</td>
<td>Olympia</td>
</tr>
<tr>
<td></td>
<td>NY</td>
<td>Albany</td>
</tr>
<tr>
<td></td>
<td>IL</td>
<td>Springfield</td>
</tr>
</tbody>
</table>

\[
\text{the } \frac{\text{NP/N}}{\lambda f. \forall x. f(x)}
\]

\[
\lambda x. \text{mountain}(x) \land \text{in}(x, \text{CO}) \quad \Rightarrow \quad \text{NP}
\]

\[
\lambda x. \text{mountain}(x) \land \text{in}(x, \text{CO})
\]
Indefinite Determiners

Indefinite determiners are select any entity from a set without a preference

\[ A : (e \rightarrow t) \rightarrow e \]

state with a mountain

\[ \lambda x.\text{state}(x) \land \text{in}(A y.\text{mountain}(y), x) \]

[Steedman 2011; Artzi and Zettlemoyer 2013b]
Indefinite Determiners

<table>
<thead>
<tr>
<th>State</th>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
<td></td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
<td></td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
<td></td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
<td></td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mountains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>Bianca</td>
</tr>
<tr>
<td>Antero</td>
</tr>
<tr>
<td>Rainier</td>
</tr>
<tr>
<td>Shasta</td>
</tr>
</tbody>
</table>

Indefinite determiners are select any entity from a set without a preference

\[ A : (e \rightarrow t) \rightarrow e \]

state with a mountain

\[ \lambda x.\text{state}(x) \land \exists y.\text{mountain}(y) \land \text{in}(y, x) \]

[Steedman 2011; Artzi and Zettlemoyer 2013b]
Indefinite Determiners

\[
\begin{align*}
\text{state} & \quad \text{with} & \quad a \\
\text{NP} & \quad \text{PP/NP} & \quad \text{NP} \\
\lambda x.\text{state}(x) & \quad \lambda x.\lambda y.\text{in}(x, y) & \quad \lambda f.\lambda x. f(x) \\
& & \quad \lambda x.\text{mountain}(x) \\
& & \quad \lambda f.\lambda y. f(y) \land (\lambda x.\text{mountain}(x), y) \\
& & \quad \lambda y.\text{state}(y) \land (\lambda x.\text{mountain}(x), y)
\end{align*}
\]
Superlatives

Superlatives select optimal entities according to a measure

the largest state

$$\arg\max (\lambda_x.\text{state}(x), \lambda_y.\text{pop}(y))$$

Min or max ... over this set ... according to this measure

{WA, AL, AK, ...}

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Capital</th>
<th>Pop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
<td>3.9</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
<td>0.4</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
<td>2.7</td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
<td>4.1</td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
<td>17.5</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
<td>11.4</td>
</tr>
</tbody>
</table>
### Superlatives

Superlatives select optimal entities according to a measure.

**the largest state**

\[ \arg\max(\lambda x.\text{state}(x), \lambda y.\text{pop}(y)) \]

Min or max ... over this set ... according to this measure.

<table>
<thead>
<tr>
<th>State</th>
<th>Capital</th>
<th>Pop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
<td>3.9</td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
<td>0.4</td>
</tr>
<tr>
<td>AZ</td>
<td>Phoenix</td>
<td>2.7</td>
</tr>
<tr>
<td>WA</td>
<td>Olympia</td>
<td>4.1</td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
<td>17.5</td>
</tr>
<tr>
<td>IL</td>
<td>Springfield</td>
<td>11.4</td>
</tr>
</tbody>
</table>

Min or max values:

<table>
<thead>
<tr>
<th>State</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>3.9</td>
</tr>
<tr>
<td>AK</td>
<td>0.4</td>
</tr>
<tr>
<td>Seattle</td>
<td>2.7</td>
</tr>
<tr>
<td>San Francisco</td>
<td>4.1</td>
</tr>
<tr>
<td>NY</td>
<td>17.5</td>
</tr>
</tbody>
</table>
## Superlatives

<table>
<thead>
<tr>
<th>State</th>
<th>Abbr.</th>
<th>Capital</th>
<th>Pop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>AL</td>
<td>Montgomery</td>
<td>3.9</td>
</tr>
<tr>
<td>AK</td>
<td>AK</td>
<td>Juneau</td>
<td>0.4</td>
</tr>
<tr>
<td>AZ</td>
<td>AZ</td>
<td>Phoenix</td>
<td>2.7</td>
</tr>
<tr>
<td>WA</td>
<td>WA</td>
<td>Olympia</td>
<td>4.1</td>
</tr>
<tr>
<td>NY</td>
<td>NY</td>
<td>Albany</td>
<td>17.5</td>
</tr>
<tr>
<td>IL</td>
<td>IL</td>
<td>Springfield</td>
<td></td>
</tr>
</tbody>
</table>

The largest state

\[
\lambda f. \text{argmax}(\lambda x. f(x), \lambda y. \text{pop}(y))
\]

\[
\frac{\lambda x. \text{state}(x)}{\text{argmax}(\lambda x. \text{state}(x), \lambda y. \text{pop}(y))}
\]
### Superlatives

<table>
<thead>
<tr>
<th>State</th>
<th>Abbr.</th>
<th>Capital</th>
<th>Pop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgo</td>
<td>3.9</td>
<td></td>
</tr>
<tr>
<td>AK</td>
<td>Junea</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>AZ</td>
<td>Phoen</td>
<td>2.7</td>
<td></td>
</tr>
<tr>
<td>WA</td>
<td>Olymp</td>
<td>4.1</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>Albany</td>
<td>17.5</td>
<td></td>
</tr>
<tr>
<td>IL</td>
<td>Spring</td>
<td>11.4</td>
<td></td>
</tr>
</tbody>
</table>

The most populated state is:

\[
\lambda g.\lambda f.\text{argmax}(\lambda x.f(x), \lambda y.g(y))
\]

\[
\lambda f.\text{argmax}(\lambda x.f(x), \lambda y.\text{pop}(y))
\]

\[
\text{argmax}(\lambda x.\text{state}(x), \lambda y.\text{pop}(y))
\]
Which mountains are in Arizona?

Represent questions as the queries that generate their answers.
Representing Questions

Which mountains are in Arizona?

```
SELECT Name FROM Mountains
WHERE State == AZ
```
Which mountains are in Arizona?

\[ \lambda x. mountain(x) \land in(x, AZ) \]

Represent questions as the queries that generate their answers

Reflects the query SQL
# Representing Questions

### State

<table>
<thead>
<tr>
<th>State</th>
<th>Abbr.</th>
<th>Capital</th>
<th>Pop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Montgomery</td>
<td>3.9</td>
<td></td>
</tr>
<tr>
<td>AK</td>
<td>Juneau</td>
<td>0.4</td>
<td></td>
</tr>
</tbody>
</table>

### Border

<table>
<thead>
<tr>
<th>Border</th>
<th>State1</th>
<th>State2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>WA</td>
<td>ID</td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>OR</td>
<td></td>
</tr>
</tbody>
</table>

### Mountains

<table>
<thead>
<tr>
<th>Mountains</th>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bianca</td>
<td>CO</td>
</tr>
<tr>
<td></td>
<td>Antero</td>
<td>CO</td>
</tr>
</tbody>
</table>

---

**How many states border California?**

\[
\text{count}(\lambda x. \text{state}(x) \land \text{border}(x, CA))
\]

**Represent questions as the queries that generate their answers**

**Reflects the query SQL**
More Reading about Modeling

Type-Logical Semantics
by Bob Carpenter

[Carpenter 1997]
Today

Parsing: Combinatory Categorial Grammars
Learning: Unified learning algorithm
Modeling: Best practices for semantics design
[fin]