Semantic Parsing with Combinatory Categorial Grammars
Yoav Artzi, Nicholas FitzGerald and Luke Zettlemoyer
University of Washington

ACL 2013 Tutorial
Sofia, Bulgaria

Website http://yoavartzi.com/tutorial
Language to Meaning

More informative
Language to Meaning

Information Extraction

Recover information about pre-specified relations and entities

Example Task

Relation Extraction

is_a(\textit{OBAMA, PRESIDENT})
Language to Meaning

Broad-coverage Semantics

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

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

Semantic Parsing

Recover complete meaning representation

More informative

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

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

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

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

\[ f : \text{sentence} \rightarrow \text{logical form} \]
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]
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: Combinatory 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
• Semantic modeling for:
  - Querying databases
  - Referring to physical objects
  - Executing instructions
UW 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
Lambda Calculus

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

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

[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

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

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

Literals

- Represent function application

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

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

Predicate: Logical expression

Arguments: List of logical expressions
Lambda Calculus

Lambda Terms

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

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

\[ \lambda x.\lambda y.\text{located\_in}(x, y) \]
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

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

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

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

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

for \( Q \in \{ \nu, \land, \lor, \forall \} \)
\[ \lambda x. \text{flight}(x) \land \text{to}(x, \text{move}) \]
\[ \lambda x. \text{flight}(x) \land \text{to}(x, \text{NYC}) \]
\[ \lambda x. \text{NYC}(x) \land x(\text{to, move}) \]
\[ \lambda x. \text{flight}(x) \land \text{to}(x, \text{move}) \]

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

\[ \times \lambda x. \text{NYC}(x) \land x(\text{to, move}) \]
Simply Typed Lambda Calculus

- Like lambda calculus
- But, typed

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

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

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

[Church 1940]
Lambda Calculus

Typing

- Simple types
- Complex types

\[ \langle e, t \rangle \]

\[ \ll \langle e, t \rangle, e \rr \]
Lambda Calculus

Typing

- Simple types
- Complex types

$t$ Truth-value
$e$ Entity

$\langle e, t \rangle$

Type constructor

Domain
Range
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

• Hierarchical typing system
Simply Typed Lambda Calculus

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

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>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>
</tbody>
</table>

<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>
<tr>
<td>CA</td>
<td>NV</td>
<td></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>
<tr>
<td>Wrangel</td>
<td>AK</td>
<td></td>
</tr>
<tr>
<td>Sill</td>
<td>CA</td>
<td></td>
</tr>
<tr>
<td>Bona</td>
<td>AK</td>
<td></td>
</tr>
<tr>
<td>Elbert</td>
<td>CO</td>
<td></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 seventh 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

[Conversational Continues]

[Artzi and Zettlemoyer 2011]
Capturing Meaning with Lambda Calculus

go to the chair and turn right

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

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

- Flexible representation
- Can capture full complexity of natural language

More on modeling meaning later
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

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

[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

Syntax $ADJ : \lambda x.\text{fun}(x)$ Semantics

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

Syntax

$ADJ : \lambda x. fun(x)$

$(S\backslash NP)/ADJ : \lambda f. \lambda x. f(x)$

$NP : CCG$

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

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

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

\[ NP : CCG \]

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

fun \vdash ADJ \colon \lambda x. 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

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

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

\[ \text{CCG} \vdash NP : \text{CCG} \]

- 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

\[ B : g \]
\[ A \backslash B : f \implies A : f(g) \quad (\langle) \]
\[ A / B : f \quad B : g \implies A : f(g) \quad (\rangle) \]

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

<table>
<thead>
<tr>
<th>CCG</th>
<th>is</th>
<th>fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>$S \backslash NP/ADJ$</td>
<td>$ADJ$</td>
</tr>
<tr>
<td>CCG</td>
<td>$\lambda f. \lambda x. f(x)$</td>
<td>$\lambda x. fun(x)$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>CCG</th>
<th>is</th>
<th>fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>S\NP/ADJ</td>
<td>ADJ</td>
</tr>
<tr>
<td>CCG</td>
<td>λf.λx.f(x)</td>
<td>λx.fun(x)</td>
</tr>
</tbody>
</table>

Combine categories using operators

A/B : f    B : g ⇒ A : f(g) (>)
Parsing with CCGs

Combine categories using operators

\[ B : g \quad A \backslash B : f \Rightarrow A : f(g) \quad (\langle \rangle) \]
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

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

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

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

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

- 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>$N\setminus N : \lambda f.\lambda x.f(x) \land g(x)$</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>$S/S : \lambda f.\lambda e.f(e) \land g(e)$</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></td>
<td></td>
</tr>
</tbody>
</table>

| ADJ: $\lambda x.g(x)$ | $N/N: \lambda f.\lambda x.f(x) \wedge g(x)$ |
| PP: $\lambda x.g(x)$ | $N\backslash N: \lambda f.\lambda x.f(x) \wedge g(x)$ |
| AP: $\lambda e.g(e)$ | $S\backslash S: \lambda f.\lambda e.f(e) \wedge g(e)$ |
| AP: $\lambda e.g(e)$ | $\textbf{S}/S: \lambda f.\lambda e.f(e) \wedge g(e)$ |

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

and \( \vdash C : conj \)

or \( \vdash C : disj \)

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

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

Coordinate composed adjectives

\[
\begin{array}{cccc}
\text{square} & \text{blue} & \text{or} & \text{round} \\
\text{ADJ} & \text{ADJ} & \text{C} & \text{ADJ} \\
\lambda x.\text{square}(x) & \lambda x.\text{blue}(x) & \text{disj} & \lambda x.\text{round}(x) \\
\text{N/N} & \text{N/N} & \text{N/N} & \text{N/N} \\
\lambda f.\lambda x.f(x) \land \text{square}(x) & \lambda f.\lambda x.f(x) \land \text{blue}(x) & \text{disj} & \lambda f.\lambda x.f(x) \land \text{round}(x) \\
\text{N/N} & \text{N/N} & \text{N/N} & \text{N/N} \\
\lambda f.\lambda x.f(x) \land \text{square}(x) \land \text{blue}(x) & \lambda f.\lambda x.f(x) \land \text{blue}(x) & \text{N/N} & \lambda f.\lambda x.f(x) \land \text{round}(x) \land \text{yellow}(x) \\
\text{N/N} & \text{N/N} & \text{N/N} & \text{N/N} \\
\lambda f.\lambda x.f(x) \land \text{square}(x) \land \text{blue}(x) & \lambda f.\lambda x.f(x) \land \text{blue}(x) & \text{N/N} & \lambda f.\lambda x.f(x) \land \text{round}(x) \land \text{yellow}(x) \\
\text{N/N} & \text{N/N} & \text{N/N} & \text{N/N} \\
\lambda f.\lambda x.f(x) \land (((\text{square}(x) \land \text{blue}(x))) \lor (\text{round}(x) \land \text{yellow}(x))) & \lambda f.\lambda x.f(x) \land \text{blue}(x) & \text{N/N} & \lambda f.\lambda x.f(x) \land \text{round}(x) \land \text{yellow}(x) \\
\text{N/N} & \text{N/N} & \text{N/N} & \text{N/N} \\
\end{array}
\]
Apply coordinated adjectives to noun

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

\[ \mathcal{X} \xrightarrow{\text{CCG}} \text{is} \xrightarrow{\lambda f. \lambda x. f(x)} \text{fun} \xrightarrow{\lambda x. \text{fun}(x)} \mathcal{Y} \]

\[ \mathcal{Y} \xrightarrow{\text{NP}} \mathcal{X} \]

---

**Lexical Ambiguity** + **Many parsing decisions** → **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., $ADJ : \lambda x.\ function(x)$

- 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]
The Lexicon Problem

• Key component of CCG
• Same words often paired with many different categories
• Difficult to learn with limited data
Factored Lexicons

the house dog

the dog of the house
\[ \nu x.\text{dog}(x) \land of(x, \nu y.\text{house}(y)) \]

the garden dog
\[ \nu x.\text{dog}(x) \land of(x, \nu y.\text{garden}(y)) \]

• Lexical entries share information

• Decomposition of entries can lead to more compact lexicons

[Kwiatkowski et al. 2011]
Factored Lexicons

the house dog

the dog of the house

the garden dog

• Lexical entries share information

• Decomposition of entries can lead to more compact lexicons
Factored Lexicons

the house dog

\[ \text{house} \vdash ADJ : \lambda x.\text{of}(x, \text{\textcolor{blue}{house}}(y)) \]

\[ \lambda x.\text{dog}(x) \land \text{of}(x, \text{\textcolor{blue}{house}}(y)) \]

the dog of the house

\[ \text{house} \vdash N : \lambda x.\text{\textcolor{blue}{house}}(x) \]

\[ \lambda x.\text{dog}(x) \land \text{of}(x, \text{\textcolor{blue}{house}}(y)) \]

the garden dog

\[ \text{garden} \vdash ADJ : \lambda x.\text{of}(x, \text{\textcolor{blue}{garden}}(y)) \]

\[ \lambda x.\text{dog}(x) \land \text{of}(x, \text{\textcolor{blue}{garden}}(y)) \]

• Lexical entries share information

• Decomposition of entries can lead to more compact lexicons
Factored Lexicons

the house dog

the dog of the house

the garden dog

• Lexical entries share information

• Decomposition of entries can lead to more compact lexicons
Factored Lexicons

- **house**: $ADJ : \lambda x. of(x, uy.house(y))$
- **house**: $\lambda x. house(x)$
- **garden**: $ADJ : \lambda x. of(x, uy.garden(y))$

**Lexemes**

- (garden, \{garden\})
- (house, \{house\})

**Templates**

\[
\lambda(\omega, \{v_i\}_{1}^{n}). \\
[\omega \vdash ADJ : \lambda x. of(x, uy.v_{1}(y))] \\
\lambda(\omega, \{v_i\}_{1}^{n}). \\
[\omega \vdash N : \lambda x. v_{1}(x)]
\]
**Factored Lexicons**

<table>
<thead>
<tr>
<th>Templates</th>
<th>Lexemes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda(\omega, {v_i}_1^n). \</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( [\omega \vdash ADJ : \lambda x.\text{of}(x, \nu y.v_1(y))] )</td>
</tr>
<tr>
<td>( \lambda(\omega, {v_i}_1^n). \</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( [\omega \vdash N : \lambda x.v_1(x)] )</td>
</tr>
<tr>
<td>( \text{Lexemes} )</td>
<td>( \text{Lexemes} )</td>
</tr>
<tr>
<td></td>
<td>( \text{(garden, {garden})} )</td>
</tr>
<tr>
<td></td>
<td>( \text{(house, {house})} )</td>
</tr>
</tbody>
</table>

- Capture systematic variations in word usage
- Each variation can then be applied to compact units of lexical meaning
- Model word meaning
- Abstracts the compositional nature of the word
Factored Lexicons

\[(\text{garden}, \{\text{garden}\})\]

\[
\lambda(\omega, \{v_i\}^n_1). \\
[\omega \vdash N : \lambda x. v_1(x)]
\]

\[
\omega \leftarrow \text{garden} \\
v_1 \leftarrow \text{garden}
\]

\[
\text{garden} \vdash N : \lambda x. \text{garden}(x)
\]
# Factored Lexicons

<table>
<thead>
<tr>
<th>Original Lexicon</th>
<th>Factored Lexicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>flight ⊢ ( S</td>
<td>NP : \lambda x.\text{flight}(x) )</td>
</tr>
<tr>
<td>flight ⊢ ( S</td>
<td>NP/(S</td>
</tr>
<tr>
<td>flight ⊢ ( S</td>
<td>NP\backslash(S</td>
</tr>
<tr>
<td>ground transport ⊢ ( S</td>
<td>NP : \lambda x.\text{trans}(x) )</td>
</tr>
<tr>
<td>ground transport ⊢ ( S</td>
<td>NP/(S</td>
</tr>
<tr>
<td>ground transport ⊢ ( S</td>
<td>NP\backslash(S</td>
</tr>
</tbody>
</table>
Factoring a Lexical Entry

house ⊢ ADJ : λx.of(x, vy.house(y))

Partial factoring
(house, \{house\})
\(\lambda(\omega, \{v_i\}_1^n).[\omega ⊢ ADJ : \lambda x.of(x, vy.v_1(y))]\)

Partial factoring
(house, \{of\})
\(\lambda(\omega, \{v_i\}_1^n).[\omega ⊢ ADJ : \lambda x.v_1(x, vy.house(y))]\)

Maximal factoring
(house, \{of, house\})
\(\lambda(\omega, \{v_i\}_1^n).[\omega ⊢ ADJ : \lambda x.v_1(x, vy.v_2(y))]\)
• Lambda calculus
• Parsing with Combinatory Categorial Grammars
• Linear CCGs
• Factored lexicons
Learning

• What kind of data/supervision we can use?
• What do we need to learn?
show me flights to Boston

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

\[
\frac{PP}{\lambda x. \text{to}(x, \text{BOSTON})} \\
\frac{\text{N}\text{N}}{\lambda f. \lambda x. \text{flight}(x) \land \text{to}(x, \text{BOSTON})} \\
\frac{\text{N}}{\lambda x. \text{flight}(x) \land \text{to}(x, \text{BOSTON})} \\
\end{array}
\]

\[
\begin{array}{c}
S \\
\lambda x. \text{flight}(x) \land \text{to}(x, \text{BOSTON})
\end{array}
\]
Learning CCG

\[
\begin{align*}
\text{show me} & \quad \text{flights} & \quad \text{to} & \quad \text{Boston} \\
S/N & \quad \lambda f. f & \quad N & \quad \lambda x. flight(x) & \quad PP/NP & \quad \lambda y. \lambda x. to(x, y) & \quad NP & \quad BOSTON \\
\lambda x. flight(x) & \quad \lambda x. to(x, BOSTON) & \quad N \setminus N & \quad \lambda f. \lambda x. f(x) \land to(x, BOSTON) & \quad < \\
\lambda x. flight(x) \land to(x, BOSTON) & \quad S & \quad \lambda x. flight(x) \land to(x, BOSTON)
\end{align*}
\]
Learning CCG

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

\[
\begin{array}{c}
\text{PP} \\
\lambda x.\text{to}(x, \text{BOSTON}) \\
\text{N} \setminus \text{N} \\
\lambda f.\lambda x. f(x) \land \text{to}(x, \text{BOSTON}) \\
\text{N} \\
\lambda x.\text{flight}(x) \land \text{to}(x, \text{BOSTON}) \\
S \\
\lambda x.\text{flight}(x) \land \text{to}(x, \text{BOSTON})
\end{array}
\]

Lexicon

Predefined

Combinators
Learning CCG

show me flights to Boston

\[
\begin{array}{c}
S/N \\
\lambda f. f
\end{array}
\quad
\begin{array}{c}
N \\
\lambda x. \text{flight}(x)
\end{array}
\quad
\begin{array}{c}
PP/NP \\
\lambda y. \lambda x. \text{to}(x, y)
\end{array}
\quad
\begin{array}{c}
NP \\
\text{BOSTON}
\end{array}
\]

\[
\begin{array}{c}
\text{PP} \\
\lambda x. \text{to}(x, \text{BOSTON})
\end{array}
\quad
\begin{array}{c}
\text{N}\backslash\text{N} \\
\lambda f. \lambda x. f(x) \land \text{to}(x, \text{BOSTON})
\end{array}
\quad
\begin{array}{c}
\text{N} \\
\lambda x. \text{flight}(x) \land \text{to}(x, \text{BOSTON})
\end{array}
\quad
\begin{array}{c}
\text{S} \\
\lambda x. \text{flight}(x) \land \text{to}(x, \text{BOSTON})
\end{array}
\]

Lexicon

Combinators

Predefined

\(W\)
Supervised Data

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

$\text{BOSTON} > PP$

$\lambda x. \text{to}(x, \text{BOSTON})$

$\text{BOSTON} > PP$

$\lambda f. \lambda x. f(x) \land \text{to}(x, \text{BOSTON})$

$\text{BOSTON} > N \setminus N$

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

$\text{BOSTON} > N$

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

$\text{BOSTON} > S$

$\lambda x. \text{flight}(x) \land \text{to}(x, \text{BOSTON})$
**Supervised Data**

<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>$\lambda x.flight(x)$</td>
</tr>
<tr>
<td>$PP/NP$</td>
</tr>
<tr>
<td>$\lambda y.\lambda x.to(x, y)$</td>
</tr>
<tr>
<td>$NP$</td>
</tr>
<tr>
<td>$BOSTON$</td>
</tr>
</tbody>
</table>

$\lambda f.\lambda x.f(x) \land to(x, BOSTON)$

$\lambda x.flight(x) \land to(x, BOSTON)$

$\lambda x.flight(x)$

$\lambda x.to(x, BOSTON)$
Supervised Data

Supervised learning is done from pairs of sentences and logical forms

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

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

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

[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*

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

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

[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 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]
**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: \( \lambda x. \text{from}(fl,x) \))

**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: \( \lambda x. \text{date}(fl,x) \land \text{to}(fl, BOS) \))

**USER** june seventh

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

Data: \( \{(x_i, y_i) : i = 1 \ldots n\} \)

For \( t = 1 \ldots T \):

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

\[ y^* \leftarrow \arg \max_y \langle \theta, \Phi(x_i, y) \rangle \]

If \( y^* \neq y_i \):

\[ \theta \leftarrow \theta + \Phi(x_i, y_i) - \Phi(x_i, y^*) \]

[iterate epochs]

[iterate examples]

[predict]

[check]

[update]

[Collins 2002]
One Derivation of the Perceptron

Log-linear model: \( p(y|x) = \frac{e^{w \cdot f(x,y)}}{\sum_{y'} e^{w \cdot f(x,y')}} \)

Step 1: Differentiate, to maximize data log-likelihood

\[
\text{update} = \sum_i f(x_i, y_i) - E_{p(y|x_i)} f(x_i, y)
\]

Step 2: Use online, stochastic gradient updates, for example \( i \):

\[
\text{update}_i = f(x_i, y_i) - E_{p(y|x_i)} f(x_i, y)
\]

Step 3: Replace expectations with maxes (Viterbi approx.)

\[
\text{update}_i = f(x_i, y_i) - f(x_i, y^*) \quad \text{where} \quad y^* = \arg \max_y w \cdot f(x_i, y)
\]
The Perceptron with Hidden Variables

Log-linear model: 
\[ p(y|x) = \sum_h p(y, h|x) \quad p(y, h|x) = \frac{e^{w \cdot f(x, h, y)}}{\sum_{y', h'} e^{w \cdot f(x, h', y')}} \]

Step 1: Differentiate marginal, to maximize data log-likelihood

\[ \text{update} = \sum_i E_{p(h|y_i, x_i)}[f(x_i, h, y_i)] - E_{p(y, h|x_i)}[f(x_i, h, y)] \]

Step 2: Use online, stochastic gradient updates, for example \( i \):

\[ \text{update}_i = E_{p(y_i, h|x_i)}[f(x_i, h, y_i)] - E_{p(y, h|x_i)}[f(x_i, h, y)] \]

Step 3: Replace expectations with maxes (Viterbi approx.)

\[ \text{update}_i = f(x_i, h', y_i) - f(x_i, h^*, y^*) \quad \text{where} \]

\[ y^*, h^* = \arg \max_{y, h} w \cdot f(x_i, h, y) \quad \text{and} \quad h' = \arg \max_h w \cdot f(x_i, h, y_i) \]
Hidden Variable Perceptron

Data: \( \{(x_i, y_i) : i = 1 \ldots n\} \)

For \( t = 1 \ldots T \): [iterate epochs]

For \( i = 1 \ldots n \): [iterate examples]

\[ y^*, h^* \leftarrow \text{arg max}_{y,h} \langle \theta, \Phi(x_i, h, y) \rangle \] [predict]

If \( y^* \neq y_i \): [check]

\[ h' \leftarrow \text{arg max}_h \langle \theta, \Phi(x_i, h, y_i) \rangle \] [predict hidden]

\[ \theta \leftarrow \theta + \Phi(x_i, h', y_i) - \Phi(x_i, h^*, y^*) \] [update]

[Liang et al. 2006; Zettlemoyer and Collins 2007]
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

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

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

Lexical Generation Procedure

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

- Given:
  - sentence \( x \)
  - validation function \( V \)
  - 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**
- **Input:**
  \[
  \{(x_i, V_i) : i = 1 \ldots n\}
  \]
- **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 parameters and lexicon**

$\theta$ weights

$\Lambda_0$ initial lexicon
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 $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$, $\lambda \leftarrow \Lambda \cup \lambda_G$

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

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
- $x_i$ sentence
- $V_i$ validation function
- $GENLEX(x_i, V_i; \Lambda, \theta)$ 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
- $x_i$ sentence
- $V_i$ validation function
- $GENLEX(x_i, V_i; \Lambda, \theta)$ 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 $\lambda_G \leftarrow GENLEX(x_i, V_i; \Lambda, \theta)$,
   $\lambda \leftarrow \Lambda \cup \lambda_G$

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
- $x_i$ sentence
- $V_i$ validation function
- $GENLEX(x_i, V_i; \Lambda, \theta)$ lexical generation function

$\mathcal{V} : \mathcal{Y} \rightarrow \{t, f\}$

$\mathcal{Y}$ 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:
   $\lambda_i \leftarrow \bigcup_{y \in \text{MAXV}_i(Y; \theta)} \text{LEX}(y)$

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

**Step 2:** (Update parameters)

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

Get top parses

$x_i$ sentence
$k$ beam size
$GEN(x_i; \lambda)$ set of 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 $\lambda_G \leftarrow GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$, $\lambda \leftarrow \Lambda \cup \lambda_G$

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

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

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

**Step 2:** (Update parameters)

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

Get lexical entries from highest scoring valid parses

$\theta$ weights
$\mathcal{V}$ validation function
$LEX(y)$ set of lexical entries
$\phi_i(y) = \phi(x_i, y)$
$MAXV_i(Y; \theta) = \{y | y \in Y \land \mathcal{V}_i(y) \land \forall y' \in Y. \mathcal{V}_i(y) \implies \langle \theta, \Phi_i(y') \rangle \leq \langle \theta, \Phi_i(y) \rangle \}$
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)

**Step 2:** (Update parameters)

a. Set $G_i \leftarrow \text{MAX}V_i(\text{GEN}(x_i; \Lambda); \theta)$
   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:

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

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

**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 $G_i \leftarrow MAXV_i(\text{GEN}(x_i; \Lambda); \theta)$
   and $B_i \leftarrow \{ e | e \in \text{GEN}(x_i; \Lambda) \land -\mathcal{V}_i(y) \}$

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

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

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

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

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

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

$\theta$ weights

$x_i$ sentence

$\mathcal{V}_i$ validation function

$\text{GEN}(x_i; \lambda)$ set of all parses

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

$MAXV_i(Y; \theta) = \{ y | y \in Y \land \mathcal{V}_i(y) \land$ 

$\forall y' \in Y. \mathcal{V}_i(y) \implies$ 

$\langle \theta, \Phi_i(y') \rangle \leq \langle \theta, \Phi_i(y) \rangle \}$

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 $B_i \leftarrow \{e | e \in GEN(x_i; \Lambda) \land \neg \mathcal{V}_i(y)\}$

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 $\Lambda$

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 $G_i \leftarrow MAXV_i(\text{GEN}(x_i; \Lambda); \theta)$
   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:
   \[
   R_i \leftarrow \{g|g \in G_i \land \exists b \in B_i \\
   \text{s.t. } \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}
   \]
   \[
   E_i \leftarrow \{b|b \in B_i \land \exists g \in G_i \\
   \text{s.t. } \langle \theta, \Phi_i(g) - \Phi_i(b) \rangle < \gamma \Delta_i(g, b)\}
   \]

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 $\Lambda$

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

\( \mathcal{V} \) validation function

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

lexical generation function

• Two parts of the algorithm we still need to define

• Depend on the task and supervision signal
# Unified Learning Algorithm

<table>
<thead>
<tr>
<th>Supervised</th>
<th>Weakly Supervised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template-based $\textit{GENLEX}$</td>
<td>Template-based $\textit{GENLEX}$</td>
</tr>
<tr>
<td>Unification-based $\textit{GENLEX}$</td>
<td></td>
</tr>
</tbody>
</table>
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 Learning

<table>
<thead>
<tr>
<th>Supervised</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall )</td>
</tr>
<tr>
<td>Template-based <em>GENLEX</em></td>
</tr>
<tr>
<td>Unification-based <em>GENLEX</em></td>
</tr>
</tbody>
</table>
Supervised Validation Function

- Validate logical form against gold label

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

- \( y \) 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

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

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

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

[Zettlemoyer and Collins 2005]
Supervised Template-based GENLEX

Need lexemes to instantiate templates

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

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

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

...
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

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

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

Create lexemes

\begin{align*}
\text{flight} & \rightarrow (\text{flight}, \{ \text{flight} \}) \\
\text{to} & \rightarrow (\text{I want}, \{ \}) \\
\text{NYC} & \rightarrow (\text{flight to new}, \{ \text{to, NYC} \}) \\
\end{align*}
Supervised Template-based

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

I want a flight to new york

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

I want a flight from New York to Boston on Delta

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

\[
\begin{array}{cccc}
\ldots & \text{from} & \text{New York} & \text{to} & \text{Boston} & \ldots \\
\hline
\text{PP/\text{NP}} & \lambda x. \lambda y. \text{to}(y, x) & \text{\text{NP}} & \lambda x. \lambda y. \text{to}(y, x) & \text{\text{NP}} & \ldots
\end{array}
\]
Fast Parsing with Pruning

I want a flight from New York to Boston on Delta

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

... from New York to Boston...

\[
\begin{align*}
\text{from} & \quad \text{New York} & \quad \text{to} & \quad \text{Boston} \\
PP/\text{NP} & \quad \lambda x.\lambda y.\text{to}(y, x) & \quad NP & \quad \lambda x.\lambda y.\text{to}(y, x) \\
\lambda x.\lambda y.\text{to}(y, x) & \quad \text{NYC} & \quad \lambda x.\lambda y.\text{to}(y, x) & \quad \text{BOS} \\
\text{PP/\text{NP}} & \quad \lambda y.\text{to}(y, \text{NYC}) & \quad \text{PP} & \quad \lambda y.\text{to}(y, \text{BOS})
\end{align*}
\]
Fast Parsing with Pruning

I want a flight from New York to Boston on Delta

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

\[
\begin{array}{c c c}
\quad & \text{from} & \text{New York} \\
\hline
PP/NP & \lambda x. \lambda y. \text{to}(y, x) & NP \\
& NYC & \quad \\
\quad & PP \quad & \quad \\
\times & \lambda y. \text{to}(y, NYC) & \quad \\
\end{array}
\]

\[
\begin{array}{c c c}
\quad & \text{to} & \text{Boston} \\
\hline
PP/NP & \lambda x. \lambda y. \text{to}(y, x) & NP \\
& BOS & \quad \\
\quad & PP \quad & \quad \\
\quad & \lambda y. \text{to}(y, BOS) & \quad \\
\end{array}
\]
Fast Parsing with Pruning

I want a flight from New York to Boston on Delta

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

... from New York to Boston ...
# 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
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 ⊨ S : \( \lambda x. \text{flight}(x) \land \text{to}(x, BOS) \)
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)
Unification-based GENLEX

- Splitting lexical entries

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

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

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

Many possible phrase pairs × Many possible category pairs
Unification-based GENLEX

• Splitting CCG categories:

1. Split logical form $h$ to $f$ and $g$ s.t.

$$f(g) = h \quad \text{or} \quad \lambda x. f(g(x)) = h$$

$$S : \lambda x. \text{flight}(x) \land \text{to}(x, \text{BOS})$$

$$\lambda f. \lambda x. \text{flight}(x) \land f(x)$$
$$\lambda x. \text{to}(x, \text{BOS})$$
$$\lambda y. \lambda x. \text{flight}(x) \land f(x, y)$$
$$\text{BOS}$$
$$\ldots$$
Unification-based GENLEX

- Splitting CCG categories:
  1. Split logical form $h$ to $f$ and $g$ s.t.
     \[
     f(g) = h \quad \text{or} \quad \lambda x. f(g(x)) = h
     \]
  2. Infer syntax from logical form type

\[
S/(S|NP) : \lambda f. \lambda x. \text{flight}(x) \land f(x) \\
S|NP : \lambda x. \text{to}(x, \text{BOS}) \\
S : \lambda x. \text{flight}(x) \land \text{to}(x, \text{BOS}) \\
S/NP : \lambda y. \lambda x. \text{flight}(x) \land f(x, y) \\
NP : \text{BOS} \\
\ldots
\]
Unification-based GENLEX

• Split text and create all pairs

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

\[
\begin{align*}
\text{I want} & \quad S/(S|NP) : \lambda f. \lambda x. \text{flight}(x) \land f(x) \\
\text{a flight to Boston} & \quad S|NP : \lambda x. \text{to}(x, BOS) \\
\text{I want a flight} & \quad S/(S|NP) : \lambda f. \lambda x. \text{flight}(x) \land f(x) \\
\text{to Boston} & \quad S|NP : \lambda x. \text{to}(x, BOS) \\
\ldots &
\end{align*}
\]
1. Find highest scoring correct parse
2. Find split that most increases score
3. Return new lexical entries
Parameter Initialization

Compute co-occurrence (IBM Model 1) between words and logical constants

I want a flight to Boston

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

Initial score for new lexical entries: average over pairwise weights
Unification-based

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

I want a flight to Boston
Unification-based

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

I want a flight to Boston

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

1. Find highest scoring correct parse
2. Find splits that most increases score
3. Return new lexical entries
Unification-based

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

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

1. Find highest scoring correct parse
2. Find splits that most increases score
3. Return new lexical entries

\[
\begin{align*}
\text{I want a flight} & \quad \text{to Boston} \\
\frac{S/(S|NP)}{\lambda f. \lambda x. \text{flight}(x) \land f(x)} & \quad \frac{S|NP}{\lambda x. \text{to}(x, \text{BOS})} \\
\end{align*}
\]

\[
\begin{align*}
\text{I want a flight to Boston} \\
\frac{S}{\lambda x. \text{flight}(x) \land \text{to}(x, \text{BOS})}
\end{align*}
\]
Unification-based

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

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

1. Find highest scoring correct parse
2. Find splits that most increases score
3. Return new lexical entries

\[ \frac{\text{I want a flight}}{S/(S|NP)} \lambda f. \lambda x. \text{flight}(x) \land f(x) \]

\[ \frac{\text{to Boston}}{S|NP} \lambda x. \text{to}(x, BOS) \]

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

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

I want a flight to Boston

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

1. Find highest scoring correct parse
2. Find splits that most increases score
3. Return new lexical entries

\[ \frac{S/(S|NP)}{\lambda f. \lambda x. \text{flight}(x) \land f(x)} \]

\[ \frac{S|NP}{\lambda x. \text{to}(x, BOS)} \]

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

**Iteration 2**
### Unification-based 

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

I want a flight to Boston

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

1. Find highest scoring correct parse
2. Find splits that most increases score
3. Return new lexical entries

Iteration 2
I want a flight to Boston

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

1. Find highest scoring correct parse
2. Find splits that most increases score
3. Return new lexical entries

**Unification-based\]**

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

**Iteration 2**
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
90 | 72.5 | 67.5 | 90 | 75 | 70

[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>
Recap

CCGs

\[
\begin{array}{ccc}
\text{CCG} & \text{is} & \text{fun} \\
\frac{NP}{CCG} & \frac{S \backslash NP/ADJ}{\lambda f. \lambda x. f(x)} & \frac{ADJ}{\lambda x. f_{\text{un}}(x)} \\
\quad & \frac{S \backslash NP}{\lambda x. f_{\text{un}}(x)} \rightarrow \\
\quad & \frac{S}{f_{\text{un}}(CCG)} < \\
\end{array}
\]

[Steedman 1996, 2000]
Recap
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
Recap
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

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

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

## Supervised

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

## Weakly Supervised

- Template-based *GENLEX*
Weak Supervision

What is the largest state that borders Texas?

New Mexico

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

What is the largest state that borders Texas?

New Mexico

at the chair, move forward three steps past the sofa

[Clarke et al. 2010; Liang et al. 2011; Chen and Mooney 2011; Artzi and Zettlemoyer 2013b]
Weak Supervision

What is the largest state that borders Texas?

New Mexico

at the chair, move forward three steps past the sofa

Execute the logical form and observe the result
Weakly Supervised Validation Function

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

\( y \in \mathcal{Y} \) parse

\( e_i \in \mathcal{E} \) available execution result

\( \text{EXEC}(y) : \mathcal{Y} \rightarrow \mathcal{E} \)

logical form at the root of \( y \)
Weakly Supervised Validation Function

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

Domain-specific execution function: SQL query engine, navigation robot

\( y \in \mathcal{Y} \) parse
\( e_i \in \mathcal{E} \) available execution result

\( \text{EXEC}(y) : \mathcal{Y} \rightarrow \mathcal{E} \)
logistical form at the root of \( y \)
Weakly Supervised Validation Function

\[ V_i(y) = \begin{cases} 
true & \text{if } \text{EXEC}(y) \approx e_i \\
false & \text{else} 
\end{cases} \]

- \( y \in \mathcal{Y} \) parse
- \( e_i \in \mathcal{E} \) available execution result

\[ \text{EXEC}(y) : \mathcal{Y} \to \mathcal{E} \]

- logical form at the root of \( y \)

Domain-specific execution function: SQL query engine, navigation robot

Depends on supervision
Weakly Supervised Validation Function

\[ V_i(y) = \begin{cases} 
\text{true} & \text{if } \text{EXEC}(y) \approx e_i \\
\text{false} & \text{else} 
\end{cases} \]

Domain-specific execution function:
SQL query engine, navigation robot

In general: execution function is a natural part of a complete system

Depends on supervision
Weakly Supervised Validation Function

Example \( \text{EXEC}(y) \):

Robot moving in an environment
Weakly Supervised Validation Function

Example $\text{EXEC}(y)$:

Robot moving in an environment

Example supervision:

Complete Demonstration

![Diagram showing robot moving in an environment with obstacles and rewards.](image-url)
Weakly Supervised Validation Function

Example $EXEC(y)$:

Robot moving in an environment

Example supervision:

Complete Demonstration

Validate all steps
Weakly Supervised Validation Function

Example $EXEC(y)$:

Robot moving in an environment

Example supervision:

Final State

Validate only last position
Weakly Supervised

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

I want a flight to new york

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

I want

\[
\begin{align*}
\text{a flight} \\
\text{flight} \\
\text{flight to new} \\
\text{flight} \\
\text{flight to new, \{flight\}} \\
\text{(flight, \{flight\})} \\
\text{(I want, \{} \}
\end{align*}
\]

Initialize templates

\[
\begin{align*}
\text{flight} \leftarrow \text{N} : \lambda x. \text{flight}(x) \\
\text{I want} \leftarrow \text{S/NP} : \lambda x. x \\
\text{flight to new} : \text{S'\text{NP/NP}} : \lambda x. \lambda y. \text{to}(x, y) \\
\end{align*}
\]

...
Weakly Supervised

$\text{GENLEX}(x, \mathcal{V}; \Lambda, \theta)$

I want a flight to new york

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

No access to labeled logical form

Initialize templates

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

flight $\vdash N : \lambda x. \text{flight}(x)$
I want $\vdash S/\text{NP} : \lambda x. x$
flight to new $: S\backslash \text{NP/\text{NP}} : \lambda x. \lambda y. \text{to}(x, y)$
Weakly Supervised GENLEX \((x, \mathcal{V}; \Lambda, \theta)\)

I want a flight to new york

- I want
- a flight
- flight
- flight to new

... (flight, \{flight\})
(I want, \{\})
(flight to new, \{to, NYC\})
...

Use all logical constants in the system instead

Initialize templates

- flight \(\leftarrow\) \(N : \lambda x.\)flight\((x)\)
- I want \(\leftarrow\) \(S/NP : \lambda x.x\)
- flight to new \(\leftarrow\) \(S'\backslash NP/NP : \lambda x.\)\(\lambda y.to(x, y)\)
- ...

- flight, from, to,
- ground.transport, dtime, atime,
- NYC, BOS, LA, SEA, ...
Weakly Supervised

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

I want a flight to new york

I want

a flight

flight

flight to new

...

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

(I want, \{\})

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

...

\text{Initialize templates}

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

I want \vdash S/\text{NP} : \lambda x.x

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

...

Many more lexemes

Huge number of lexical entries

Use all logical constants in the system instead
Weakly Supervised

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

I want a flight to New York

...
Weakly Supervised
GENLEX \((x, \mathcal{V}; \Lambda, \theta)\)

I want a flight to new york

- flight, from, to,
- ground_transport, dtime, atime,
- NYC, BOS, LA, SEA, …

flight \(\vdash N : \lambda x.\text{flight}(x)\)
I want \(\vdash S/NP : \lambda x.x\)
flight to new : \(S\backslash NP/NP : \lambda x.\lambda y.to(x, y)\)

Parse + prune generated lexicon

Huge number of lexical entries

Intractable

Model
Weakly Supervised

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

I want a flight to new york

I want a flight
flight
flight to new

...?

(\text{flight, \{flight\}})
(I \text{ want, \{\}})
(\text{flight to new, \{to, NYC\}})
...?

**Initialize templates**

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

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

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

...
Weakly Supervised

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

- Gradually prune lexical entries using a coarse-to-fine semantic parsing algorithm
- Transition from coarse to fine defined by typing system
Coarse Ontology

\[ \text{flight}_{fl,t}, \text{from}_{fl,<loc,t>}, \text{to}_{fl,<loc,t>}, \]

\[ \text{ground\_transport}_{gt,t}, \text{dtime}_{tr,<ti,t>}, \text{atime}_{tr,<ti,t>}, \]

\[ \text{NYC}_{ci}, \text{BOS}_{ci}, \text{JFK}_{ap}, \text{LAS}_{ap}, \ldots \]
Coarse Ontology

- \textit{flight}_{fl,t} \textit{from}_{fl,<loc,t>} \textit{to}_{fl,<loc,t>},
- \textit{ground\_transport}_{gt,t} \textit{dtime}_{tr,<ti,t>} \textit{atime}_{tr,<ti,t>},
- \textit{NYC}_{ci}, \textit{BOS}_{ci}, \textit{JFK}_{ap}, \textit{LAS}_{ap}, \ldots

Generalize types

- \textit{flight}_{e,t} \textit{from}_{e,<e,t>} \textit{to}_{e,<e,t>},
- \textit{ground\_transport}_{e,t} \textit{dtime}_{e,<e,t>} \textit{atime}_{e,<e,t>},
- \textit{NYC}_{e}, \textit{BOS}_{e}, \textit{LA}_{e}, \textit{SEA}_{e}, \ldots
Coarse Ontology

\[ \text{Generalize types} \]

\[ \text{Merge identically typed constants} \]

\[ c_1^{<e,t>}, c_2^{<e,<e,t>>}, c_3^{e}, \ldots \]
Weakly Supervised
\[ \text{GENLEX}(x, \mathcal{V}; \Lambda, \theta) \]

I want a flight to new york

All possible sub-strings

\[
\begin{align*}
&c_1\langle e, t \rangle \\
&c_2\langle e, \langle e, t \rangle \rangle \\
&c_3e \\
&\ldots
\end{align*}
\]
Weakly Supervised

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

I want a flight to New York

All possible sub-strings

(I want, \{}\)

(flight, \{c1\})

(flight to new, \{c2\})

Create lexemes

...
Weakly Supervised

$\text{GENLEX}(x, \mathcal{V}; \Lambda, \theta)$

I want a flight to new york

Initialize templates
Weakly Supervised
\( \text{GENLEX} (x, \mathcal{V}; \Lambda, \theta) \)

I want a flight to new york

---

Coarse constants

- \( c_1^{e,t} \)
- \( c_2^{e,<e,t>} \)
- \( c_3 \)

Initialize templates

\( \text{flight} \vdash N : \lambda x.p_1(x) \)
\( \text{I want} \vdash S/NP : \lambda x.x \)
\( \text{flight to new} \vdash S\backslash NP/NP : \lambda x.\lambda y.p_2(x, y) \)
Weakly Supervised

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

I want a flight to new york

\[
\text{flight } \vdash N : \lambda x.c1(x)
\]

\[
\text{I want } \vdash S/NP : \lambda x.x
\]

\[
\text{flight to new } \vdash S/NP/NP : \lambda x.\lambda y.c2(x, y)
\]

...  

Keep only lexical entries that participate in complete parses, which score higher than the current best valid parse by a margin
Weakly Supervised

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

I want a flight to new york

\[
\text{flight} \vdash N : \lambda x. c_1(x) \\
\text{I want} \vdash S/NP : \lambda x.x \\
\text{flight to new} \vdash S/NP/NP : \lambda x. \lambda y. c_2(x, y)
\]

...  

Keep only lexical entries that **participate in complete parses**, which **score higher** than the current best valid parse by a margin.
Weakly Supervised

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

I want a flight to new york

flight $\vdash N : \lambda x. c_1(x)$

... Replace all coarse constants with all similarly typed constants

flight $\vdash N : \lambda x. \text{flight}(x)$
flight $\vdash N : \lambda x. \text{ground}_\text{transport}(x)$
flight $\vdash N : \lambda x. \text{nonstop}(x)$
flight $\vdash N : \lambda x. \text{connecting}(x)$

...
Weak Supervision
Requirements

• Know how to act given a logical form
• A validation function
• Templates for lexical induction
Experiments

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

Demonstration:

• Situated learning with joint inference
• Two forms of validation
• Template-based $GENLEX(x, V; \Lambda, \theta)$

[Artzi and Zettlemoyer 2013b]
Results

- **Single Sentence**
  - Final State Validation: 77.6
  - Trace Validation: 78.63

- **Sequence**
  - Final State Validation: 54.63
  - Trace Validation: 58.05

- **Logical Form**
  - Final State Validation: 44
  - Trace Validation: 51.05
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]
• Structured perceptron
• A unified learning algorithm
• Supervised learning
• Weak supervision
Show me all papers about semantic parsing

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

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>

#### Noun Phrases

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

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>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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State1</th>
<th>State2</th>
<th>Border</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>
<tr>
<td>CA</td>
<td>NV</td>
<td></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>

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>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>
</tbody>
</table>

<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>
<tr>
<td>CA</td>
<td>NV</td>
<td></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>

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?
### Querying Databases

<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>
</tbody>
</table>

<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>
<tr>
<td>CA</td>
<td>NV</td>
<td></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>

### Questions

1. **What** is the capital of Arizona?
2. **How many** states border California?
3. **What** is the largest state?
Referring to DB Entities

- **Noun phrases**: Select single DB entities
- **Prepositions & 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
The Sunshine State
FL
# Noun Phrases

## State Abbr. Capital Pop. 
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></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>
<tr>
<td>WA</td>
<td>Olympia</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>Springfield</td>
<td>11.4</td>
<td></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 
- The Sunshine State 
- FL

E-typed entities:

- Bianca 
- Antero 
- Rainier 
- Shasta

Specific entities:

- Washington 
- WA
- Florida 
- The Sunshine State 
- FL
### 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**

<table>
<thead>
<tr>
<th>NP</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
</tr>
</tbody>
</table>

The Sunshine State

<table>
<thead>
<tr>
<th>NP</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL</td>
</tr>
</tbody>
</table>
## Verb Relations

<table>
<thead>
<tr>
<th>State</th>
<th>Abbr.</th>
<th>Capital</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>

### Borders

<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>

Nevada borders California

$\text{border}(NV, CA)$

Verbs express relations between entities.
## 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

<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>

### Remarks

- Nevada **borders** California
- border(NV, CA) is **true**

Verbs express relations between entities.
Verb 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>

\[
\begin{array}{c}
NP \\
NV
\end{array}
\]

\[
\text{borders} \quad \lambda x. \lambda y. \text{border}(y, x)
\]

\[
\begin{array}{c}
\text{Nevada} \quad S \setminus \text{NP}/\text{NP} \\
\text{California} \quad S \setminus \text{NP}
\end{array}
\]

\[
\lambda y. \text{border}(y, \text{CA})
\]

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

Nouns are functions that define entity type

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

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

Nouns are functions that define entity type

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

$$\{\text{WA}, \text{AL}, \text{AK}, \ldots\}$$

$$e \rightarrow t$$

functions define sets

$$\lambda x.\text{mountain}(x)$$

$$\{\text{BIANCA}, \text{ANTERO}, \ldots\}$$

#### 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

\[
\begin{align*}
\text{state} & \quad \lambda x. \text{state}(x) \\
\text{mountain} & \quad \lambda x. \text{mountain}(x)
\end{align*}
\]
### State Capital Population

<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>

---

### 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>

---

**Prepositions**

Prepositional phrases are conjunctive modifiers.

**mountain in Colorado**
Prepositions

Prepositional phrases are conjunctive modifiers

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

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

**Prepositional phrases are conjunctive modifiers**

mountain in Colorado

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

\{ BIANCA, ANTERO \}
Prepositions

<table>
<thead>
<tr>
<th>State</th>
<th>Abbr.</th>
<th>Capital</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>

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

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

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

<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>

### 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.

`\lambda x. state(x) \land border(x, CA) \{ OR, NV, AZ \}`
**Function Words**

<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>
<tr>
<td>WA</td>
<td>Olympia</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>Springfield</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Diagram**

\[
\begin{array}{c|c|c|c}
\text{State} & \text{Abbr.} & \text{Capital} & \text{Pop.} \\
\hline
\text{AL} & \text{Montgomery} & 3.9 & \\
\text{AK} & \text{Juneau} & 0.4 & \\
\text{AZ} & \text{Phoenix} & 2.7 & \\
\text{WA} & \text{Olympia} & 4.1 & \\
\text{NY} & \text{Albany} & 17.5 & \\
\text{IL} & \text{Springfield} & & \\
\end{array}
\]

**Function Words**

\[
\begin{align*}
\text{state} & \quad \frac{N}{NV} \\
\text{that} & \quad \frac{PP/(S\backslash NP)}{\lambda f.f} \\
\text{borders} & \quad \frac{S\backslash NP/NP}{\lambda x.\lambda y.border(y, x)} \\
\text{California} & \quad \frac{NP}{CA} \\
\text{state} & \quad \frac{N}{N} \\
\lambda f.\lambda y.f(y) \land \text{border}(y, CA) & \quad \frac{N}{N} \\
\lambda x.\text{state}(x) \land (x, CA) & \quad <
\end{align*}
\]
### Function Words

<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>

#### 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>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>
<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

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

\{ \text{RAINIER} \}
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

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

the mountain in Washington

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

\{ \text{RAINIER} \} \rightarrow \text{RAINIER}
Definite Determiners

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

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

the mountain in Colorado

\( \forall x. \text{mountain}(x) \land \text{in}(x, \text{CO}) \)

\( \{ \text{BIANCA, ANTERO} \} \rightarrow ? \)
Definite Determiners

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

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

The mountain in Colorado

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

\[ \{ \text{BIANCA}, \text{ANTERO} \} \rightarrow \times \]

No information to disambiguate
## Definite Determiners

<table>
<thead>
<tr>
<th>State</th>
<th>Abbr.</th>
<th>Capital</th>
</tr>
</thead>
<tbody>
<tr>
<td>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>

**Example:**

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

\[
\frac{\lambda x. \text{mountain}(x) \land \text{in}(x, CO)}{NP} \quad \forall x. \text{mountain}(x) \land \text{in}(x, CO)
\]
### Indefinite 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>

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

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

state with a mountain

\[ \lambda x. state(x) \land in(Ay. mountain(y), x) \]

[Steedman 2011; Artzi and Zettlemoyer 2013b]
## Indefinite 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>

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

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

\[
\begin{align*}
PP & \lambda y. (\lambda x. \text{mountain}(x), y) \\
N \setminus N & \lambda f. \lambda y. f(y) \land (\lambda x. \text{mountain}(x), y) \\
N & \lambda y. \text{state}(y) \land (\lambda x. \text{mountain}(x), y)
\end{align*}
\]
Using the indefinite quantifier simplifies CCG handling of the indefinite determiner
Superlatives

Superlatives select optimal entities according to a measure.

The largest state:

$$\text{argmax} \left( \lambda_x \text{state}(x), \lambda_y \text{pop}(y) \right)$$

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

{WA, AL, AK, ...}
**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>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>
<tr>
<td>WA</td>
<td>Olympia</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>Springfield</td>
<td>11.4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>City</th>
<th>Pop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</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>MT</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>11.4</td>
</tr>
</tbody>
</table>

The largest state is determined by finding the state with the maximum population, which can be expressed as:

$$NP/N \arg\max(x.f(x), y.pop(y))$$

And the state with the largest population is:

$$\arg\max(NP \lambda x.\text{state}(x), NP \lambda y.\text{pop}(y))$$
Superlatives

<table>
<thead>
<tr>
<th>State</th>
<th>Abbr.</th>
<th>Capital</th>
<th>Pop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>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>11.4</td>
</tr>
</tbody>
</table>

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

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

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

<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>
</tbody>
</table>

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

Which mountains are in Arizona? Represent questions as the queries that generate their answers
Which mountains are in Arizona?

SELECT Name FROM Mountains
WHERE State == AZ

Represent questions as the queries that generate their answers

Reflects the query SQL
Representing Questions

Which mountains are in Arizona?

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

Represent questions as the queries that generate their answers

Reflects the query SQL
Representing Questions

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>
</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>
</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>
</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
DB Queries

So Far

• Refer to entities in a database
• Query over type of entities, order and other database properties

Next

• How does this approach hold for physical objects?
• What do we need to change? Add?
Referring to Real World Objects

[Matuszek et al. 2012a]
Referring to Real World Objects

all the arches except the green arch
Referring to Real World Objects

all the arches except the green arch
Referring to Real World Objects

the blue triangle and the green arch
Referring to Real World Objects

the blue triangle and the green arch
Plurality

\[ \text{arches} \]
\[ \lambda x. \text{arch}(x) \]
\[ \{ , , , , , \} \]
Plurality

\[ \{\text{arches}\} \]

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

the arches

\[ \forall x.\text{arch}(x) \]

\[ \times \]
Plurality

blue blocks
\( \lambda x.\text{blue}(x) \land \text{block}(x) \)
\[
\{ \text{\textbullet, \text{\textbullet}}, \} 
\]
brown block
\( \lambda x.\text{brown}(x) \land \text{block}(x) \)
\[
\{ \text{\textbullet}, \} 
\]
Plurality

- All entities are sets
- Space of entities includes singletons and sets of multiple objects
Plurality

• All entities are sets
• Space of entities includes singletons and sets of multiple objects

Cognitive evidence for sets being a primitive type

[Scontras et al. 2012]
Plurality

Plurality is a modifier and entities are defined to be sets.
Plurality is a modifier and entities are defined to be sets.

\[ \text{arch} \quad \lambda x. \text{arch}(x) \land sg(x) \]
Plurality

Plurality is a modifier and entities are defined to be sets.

\[ \text{arch} \lambda x. arch(x) \land sg(x) \]

\[
\{ \{ \text{red} \}, \{ \text{yellow} \}, \{ \text{green} \}, \{ \text{blue} \}, \text{banana} \}\]

Plurality

Plurality is a modifier and entities are defined to be sets.

arches

\( \lambda x. arch(x) \land plu(x) \)

\( \{ \{\} , \{\} , \{\} , \{\} , \{\} \} , \{\} , \{\} , \ldots \} \)
Plurality and Determiners

Definite determiner must select a single set. E.g., heuristically select the maximal set.

the arches

\( \forall x. arch(x) \land plu(x) \)

\( \{ \text{arches} \} \)
Adjectives

Adjectives are conjunctive modifiers

blue objects

\[ \lambda x. blue(x) \land obj(x) \land plu(x) \]
Adjectives

Adjectives are conjunctive modifiers

blue objects

\[ \lambda x. blue(x) \land obj(x) \land plu(x) \]

\[ \{\{\text{<>, <}}\} \]
DBs and Physical Objects

- Describe and refer to entities
- Ask about objects and relations between them
- Next: move into more dynamic scenarios
Beyond Queries

- Noun phrases: Specific entities
- Nouns: Sets of entities
- Prepositional phrases and Adjectives: Constrain sets
- Questions: Queries to generate response
Beyond Queries

Noun phrases: Specific entities
Nouns: Sets of entities
Prepositional phrases and Adjectives: Constrain sets
Questions: Queries to generate response

Works well for natural language interfaces for DBs

How can we use this approach for other domains?
Procedural Representations

- Common approach to represent instructional language
- Natural for executing commands

```
go forward along the stone hall to the intersection with a bare concrete hall

Verify(front: GRAVEL_HALL)
Travel()
Verify(side: CONCRETE_HALL)
```

[Chen and Mooney 2011]
Procedural Representations

- Common approach to represent instructional language
- Natural for executing commands

leave the room and go right

do_seq(verify(room(current_loc)),
move_to(unique_thing(\lambda x.\text{equals}(\text{distance}(x), 1))),
move_to(right_loc))

[Matuszek et al. 2012b]
Procedural Representations

- Common approach to represent instructional language
- Natural for executing commands

Click Start, point to Search, and the click For Files and Folders. In the Search for box, type “msdownld.tmp”.

`LEFT_CLICK(Start)
LEFT_CLICK(Search)
...
TYPE_INFO(Search for:, “msdownld.tmp”)`
Procedural Representations

Dissonance between structure of semantics and language

- Poor generalization of learned models
- Difficult to capture complex language
Spatial and Instructional Language

**Name objects**

<table>
<thead>
<tr>
<th>Noun phrases</th>
<th>Specific entities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nouns</td>
<td>Sets of entities</td>
</tr>
<tr>
<td>Prepositional phrases</td>
<td>Constrain sets</td>
</tr>
<tr>
<td>Adjectives</td>
<td></td>
</tr>
</tbody>
</table>

**Instructions to execute**

<table>
<thead>
<tr>
<th>Verbs</th>
<th>Davidsonian events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imperatives</td>
<td>Sets of events</td>
</tr>
</tbody>
</table>
Modeling Instructions

Describing an environment → Executing instructions

[Artzi and Zettlemoyer 2013b]
Modeling Instructions

Describing an environment

Agent

Executing instructions
Modeling Instructions

- Model actions and imperatives
- Consider how the state of the agent influences its understanding of language
Modeling Instructions

place your back against the wall of the t intersection

turn left

go forward along the pink flowered carpet hall two segments to the intersection with the brick hall
Instructional Environment

- Maps are graphs of connected positions
- Positions have properties and contain objects
Instructional Environment

- Agent can move forward, turn right and turn left
- Agent perceives clusters of positions
- Clusters capture objects
Instructional Environment

- Agent can move forward, turn right and turn left
- Agent perceives clusters of positions
- Clusters capture objects
• Agent can move forward, turn right and turn left

• Agent perceives clusters of positions

• Clusters capture objects
Instructional Environment

- Agent can move forward, turn right and turn left
- Agent perceives clusters of positions
- Clusters capture objects
### Instructional Environment

- Agent can move forward, turn right and turn left
- Agent perceives clusters of positions
- Clusters capture objects
Instructional Environment

- Refer to objects similarly to our previous domains
- “Query” the world
Grounded Resolution of Determiners

Nouns denote sets of objects

\[ \lambda x. \text{chair}(x) \]
Grounded Resolution of Determiners

Definite determiner selects a single entity

The chair

\( \forall x.\text{chair}(x) \)
Grounded Resolution of Determiners

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Definite determiner selects a single entity

\[ \lambda x. chair(x) \]

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

\{ \} \rightarrow
Grounded Resolution of Determiners

Definite determiner selects a single entity

the chair

\( \forall x.\text{chair}(x) \)
Grounded Resolution of Determiners

The definite determiner selects a single entity.

The chair

\[\text{the chair} \quad \text{\(vx.chair(x)\)}\]
Grounded Resolution of Determiners

Definite determiner selects a single entity

the chair

\( \forall x. \text{chair}(x) \)

Must disambiguate to select a single entity
Grounded Resolution of Determiners

Definite determiner depends on agent state

Agent selects a single entity

the chair

$\forall x. \text{chair}(x)$

Definite determiner depends on agent state
Grounded Resolution of Determiners

Definite determiner depends on agent state

\( \text{the chair} \)

\( \forall x.\text{chair}(x) \)

Definite determiner selects a single entity
Modeling Instructions

- Events taking place in the world
- Events refer to environment
- Implicit requests
Modeling Instructions

- Events taking place in the world
- Events refer to environment
- Implicit requests

walk forward twice
Modeling Instructions

Events taking place in the world

Events refer to environment

Implicit requests

move twice to the chair
Modeling Instructions

Events taking place in the world

Events refer to environment

Implicit requests

need to move first

at the chair, turn right
Davidsonian Event Semantics

- Actions in the world are constrained by adverbial modifiers
- The number of such modifiers is flexible

Adverbial modification is thus seen to be logically on a par with adjectival modification: what adverbial clauses modify is not verbs, but the events that certain verbs introduce.

Davidson 1969 (quoted in Maienborn et al. 2010)
Davidsonian Event Semantics

- Use event variable to represent events
- Verbs describe events like nouns describe entities
- Adverbials are conjunctive modifiers

Vincent shot Marvin in the car accidentally

\[ \exists a. \text{shot}(a, VINCENT, MARVIN) \land \neg \text{intentional}(a) \land \text{in}(a, \forall x. \text{car}(x)) \]
Neo-Davidsonian Event Semantics

Active

Vincent shot Marvin

\[ \exists a. \text{shot}(a, VINCENT, MARVIN) \]
Neo-Davidsonian Event Semantics

**Active**

Vincent shot Marvin

\[ \exists a.\text{shot}(a, VINCENT, MARVIN) \]

**Passive**

Marvin was shot by Vincent

[Parsons 1990]
Neo-Davidsonian Event Semantics

Active

Vincent shot Marvin

\[ \exists a. \text{shot}(a, VINCENT, MARVIN) \]

Passive

Marvin was shot (by Vincent)

Agent optional in passive

[Parsons 1990]
Neo-Davidsonian Event Semantics

Active
Vincent shot Marvin
\( \exists a. shot(a, VINCENT, MARVIN) \)

Passive
Marvin was shot (by Vincent)
\( \exists a. shot(a, MARVIN) \)

Agent optional in passive

[Parsons 1990]
Neo-Davidsonian Event Semantics

Active

Vincent shot Marvin

\[ \exists a. \text{shot}(a, \text{VINCENT}, \text{MARVIN}) \]

Passive

Marvin was shot (by Vincent)

\[ \exists a. \text{shot}(a, \text{MARVIN}) \]

Can we represent such distinctions without requiring different arity predicates?

[Parsons 1990]
Neo-Davidsonian Event Semantics

- Separation between semantic and syntactic roles
- Thematic roles captured by conjunctive predicates

Vincent shot Marvin

$$\exists a. \text{shot}(a, \text{VINCENT}, \text{MARVIN})$$

$$\exists a. \text{shot}(a) \land \text{agent}(a, \text{VINCENT}) \land \text{patient}(a, \text{MARVIN})$$

[Parsons 1990]
Neo-Davidsonian Event Semantics

Vincent shot Marvin in the car accidentally

$$\exists a. \text{shot}(a) \land \text{agent}(a, VINCENT) \land$$
$$\text{patient}(a, MARVIN) \land \text{in}(a, \forall x. \text{car}(x)) \land \neg \text{intentional}(a)$$

- Decomposition to conjunctive modifiers makes incremental interpretation simpler
- Shallow semantic structures: no need to modify deeply embedded variables

[Parsons 1990]
Neo-Davidsonian Event Semantics

\exists a. \text{shot}(a) \land \text{agent}(a, VINCENT) \land \\
\text{patient}(a, MARVIN) \land \text{in}(a, \forall x. \text{car}(x)) \land \neg \text{intentional}(a)

Without events:

\text{shot}(VINCENT, MARVIN, \forall x. \text{car}(x), INTENTIONAL)

- Decomposition to conjunctive modifiers makes incremental interpretation simpler
- Shallow semantic structures: no need to modify deeply embedded variables

[Parsons 1990]
Representing Imperatives

move forward past the sofa to the chair
Representing Imperatives

move forward past the sofa to the chair
Representing Imperatives

**Type**: move

**Direction**: forward

**Intermediate position**: past the sofa

**Final position**: to the chair
Representing Imperatives

• Imperatives define actions to be executed
• Constrained by adverbials
• Similar to how nouns are defined
Representing Imperatives

- Imperatives are sets of actions
- Just like nouns: functions from events to truth

\[ f : ev \rightarrow t \]
Representing Imperatives

Type: move
Direction: forward
Intermediate position: past the sofa
Final position: to the chair

Given a set, what do we actually execute?
Representing Imperatives

- Need to select a single action and execute it
- Reasonable solution: select simplest/shortest
Modeling Instructions

- Imperatives are sets of events
- Events are sequences of identical actions

\[
\lambda a. \text{move}(a)
\]

\[
\{ \text{chair}, \text{chair}, \text{chair} \}
\]
Modeling Instructions

- Imperatives are sets of events
- Events are sequences of identical actions

\[ \lambda a. \text{move}(a) \]

Disambiguate by preferring shorter sequences
Events can be modified by adverbials

\[ \lambda a. \text{move}(a) \land \text{len}(a, 2) \]
Events can be modified by adverbials

\[
\text{go to the chair} \quad \lambda a. \text{move}(a) \land \\
\quad \text{to}(a, \forall x. \text{chair}(x))
\]
Modeling Instructions

Treatment of events and their adverbials is similar to nouns and prepositional phrases.
Modeling Instructions

Dynamic Models

Implicit Actions
World model changes during execution

move until you reach the chair

\[ \lambda a. \text{move}(a) \wedge \]
\[ \text{post}(a, \text{intersect}(\exists x. \text{chair}(x), \text{you})) \]
Dynamic Models

World model changes during execution

move until you reach the chair

\( \lambda a. \text{move}(a) \land \text{post}(a, \text{intersect}(x.\text{chair}(x), \text{you})) \)
Dynamic Models

World model changes during execution

move until you reach the chair

\[ \lambda a. move(a) \land \\
post(a, intersect(ix.chair(x), you)) \]

Never intersects
Dynamic Models

World model changes during execution

move until you reach the chair

\[ \lambda a. \text{move}(a) \land \text{post}(a, \text{intersect}(\forall x. \text{chair}(x), \text{you})) \]

Update model to reflect state change
Dynamic Models

World model changes during execution

move until you reach the chair

\[ \lambda_a.\text{move}(a) \land \text{post}(a, \text{intersect}(\forall x.\text{chair}(x), \text{you})) \]

Update

Update model to reflect state change
Consider action assignments with prefixed implicit actions

at the chair, turn left

\[ \lambda a.\text{turn}(a) \land \text{dir}(a, \text{left}) \land \text{pre}(a, \text{intersect}(\forall x.\text{chair}(x), \text{you})) \]
Implicit Actions

Consider action assignments with prefixed implicit actions at the chair, turn left

\[ \lambda a. \text{turn}(a) \land \text{dir}(a, \text{left}) \land \text{pre}(a, \text{intersect}(\forall x. \text{chair}(x), \text{you})) \]
Consider action assignments with prefixed implicit actions

at the chair, turn left

\[ \lambda a. \text{turn}(a) \land \text{dir}(a, \text{left}) \land \pre(a, \text{intersect}(\forall x. \text{chair}(x), \text{you})) \]

Implicit actions
Experiments

Instruction:

at the chair, move forward three steps past the sofa

Demonstration:

- Situated learning with joint inference
- Two forms of validation
- Template-based $GENLEX(x, V; \Lambda, \theta)$

[Artzi and Zettlemoyer 2013b]
Results

SAIL Corpus - Cross Validation

[Artzi and Zettlemoyer 2013b]
More Reading about Modeling

Type-Logical Semantics by Bob Carpenter
Today

- Parsing: Combinatory Categorial Grammars
- Learning: Unified learning algorithm
- Modeling: Best practices for semantics design
Looking Forward
Looking Forward: Scale

**Goal**
Answer any question posed to large, community authored databases

**Challenges**
- Large domains
- Scalable algorithms
- Unseen words and concepts

**See**
Cai and Yates 2013a, 2013b

What are the neighborhoods in New York City?
\[ \lambda x . \text{neighborhoods}(\text{new\_york}, x) \]

How many countries use the rupee?
\[ \text{count}(x). \text{countries\_used}(\text{rupee}, x) \]

How many Peabody Award winners are there?
\[ \text{count}(x). \exists y . \text{award\_honor}(y) \land \text{award\_winner}(y, x) \land \text{award}(y, \text{peabody\_award}) \]
Looking Forward: Code

Goal

Program using natural language
- Data
- Complex intent
- Complex output

Challenges

Kushman and Barzilay 2013; Lei et al. 2013

See

(a) Text Specification:
The input contains a single integer T that indicates the number of test cases. Then follow the T cases. Each test case begins with a line contains an integer N, representing the size of wall. The next N lines represent the original wall. Each line contains N characters. The j-th character of the i-th line figures out the color ...

(b) Specification Tree:

the input

a single integer T

an integer N

the next N lines

N characters
test cases

c

(c) Two Program Input Examples:

1
10
YWWYWWWW
YWWYWWWW
YWWYWWWW
...
WWWWWWWW

2
1
Y
5
YWWWW
...
WYYYY

<table>
<thead>
<tr>
<th>Text Description</th>
<th>Regular Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>three letter word starting with <code>X</code></td>
<td>\X[A-Za-z]{2}\b</td>
</tr>
</tbody>
</table>
Looking Forward: Context

**Goal**
Understanding how sentence meaning varies with context

**Challenges**
- Data
- Linguistics: co-ref, ellipsis, etc.

**See**
Miller et al. 1996; Zettlemoyer and Collins 2009; Artzi and Zettlemoyer 2013

---

Example #1:
(a) show me the flights from boston to philly
   \[ \lambda x. \text{flight}(x) \land \text{from}(x, \text{bos}) \land \text{to}(x, \phi) \]
(b) show me the ones that leave in the morning
   \[ \lambda x. \text{flight}(x) \land \text{from}(x, \text{bos}) \land \text{to}(x, \phi) \land \text{during}(x, \text{morning}) \]
(c) what kind of plane is used on these flights
   \[ \lambda y. \exists x. \text{flight}(x) \land \text{from}(x, \text{bos}) \land \text{to}(x, \phi) \land \text{during}(x, \text{morning}) \land \text{aircraft}(x) = y \]

Example #2:
(a) show me flights from milwaukee to orlando
   \[ \lambda x. \text{flight}(x) \land \text{from}(x, \text{mil}) \land \text{to}(x, \text{orl}) \]
(b) cheapest
   \[ \arg \min (\lambda x. \text{flight}(x) \land \text{from}(x, \text{mil}) \land \text{to}(x, \text{orl}), \lambda y. \text{fare}(y)) \]
(c) departing wednesday after 5 o’clock
   \[ \arg \min (\lambda x. \text{flight}(x) \land \text{from}(x, \text{mil}) \land \text{to}(x, \text{orl}) \land \text{day}(x, \text{wed}) \land \text{depart}(x) > 1700, \lambda y. \text{fare}(y)) \]
Looking Forward: Sensors

Goal
Integrate semantic parsing with rich sensing on real robots

Challenges
- Data
- Managing uncertainty
- Interactive learning

See

Move the pallet from the truck.
Remove the pallet from the back of the truck.
Offload the metal crate from the truck.
UW 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]
[fin]
Supplementary Material
Function Composition

\[ g_{\langle \alpha, \beta \rangle} = \lambda x. G \]
\[ f_{\langle \beta, \gamma \rangle} = \lambda y. F \]
\[ g(A) = (\lambda x. G)(A) = G[x := A] \]
\[ f(g(A)) = (\lambda y. F)(G[x := A]) = F[y := G[x := A]] \]
\[ \lambda x. f(g(A))[A := x] = \lambda x. F[y := G[x := A]][A := x] = \lambda x. F[y := G] = (f \cdot g)_{\langle \alpha, \gamma \rangle} \]
References


