Abridged

Semantic Parsing with Combinatory Categorial Grammars

Yoav Artzi
University of Washington

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

Language to Meaning

More informative

Language to Meaning

Information Extraction

Recover information about pre-specified relations and entities

More informative

Example Task

Relation Extraction

$$is\_a(OBAMA, PRESIDENT)$$

Language to Meaning

Broad-coverage Semantics

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

More informative

Example Task

Summarization


The New York Times

OBAMA

RACIAL BARRIER FALLS IN DECISIVE VICTORY

Democrats in Congress Strengthen Grip

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

Language to Meaning

Semantic
Parsing

Recover **complete** meaning representation

More informative

Example Task

Database Query

What states border Texas?

Oklahoma
New Mexico
Arkansas
Louisiana

Language to Meaning

Semantic Parsing

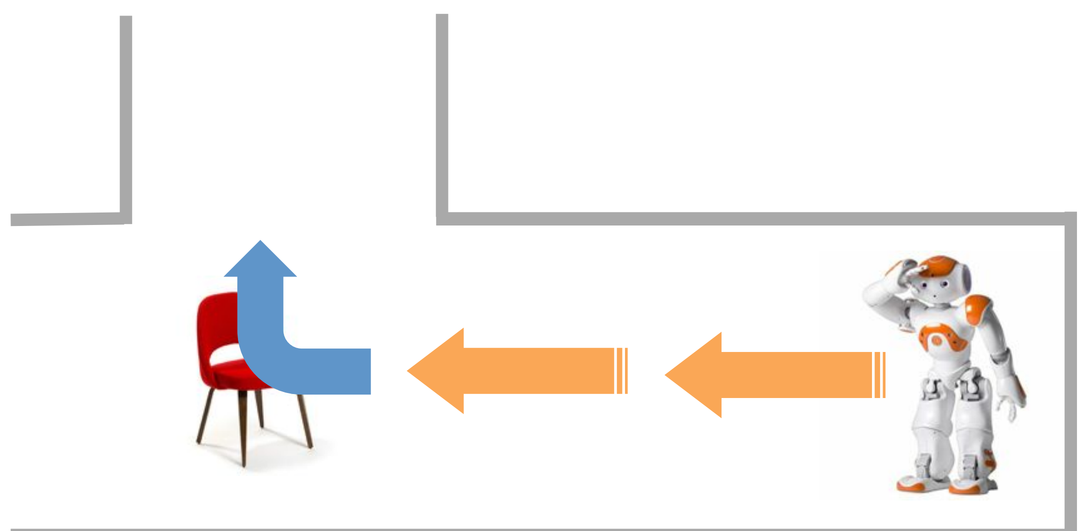
Recover **complete** meaning representation

More informative

Example Task

Instructing a Robot

at the chair,
turn right



Language to Meaning

<u>Complete</u> meaning is sufficient to complete the task

- Convert to database query to get the answer
- Allow a robot to do planning

Language to Meaning

at the chair, move forward three steps past the sofa

$$\lambda a.pre(a, \iota x.chair(x)) \wedge move(a) \wedge len(a, 3) \wedge$$
$$dir(a, forward) \wedge past(a, \iota y.sofa(y))$$

Language to Meaning

Semantic Parsing

Recover **complete** meaning representation

More informative

at the chair, move forward three steps past the sofa

$$\lambda a.pre(a, \iota x.chair(x)) \wedge move(a) \wedge len(a, 3) \wedge$$
$$dir(a, forward) \wedge past(a, \iota y.sofa(y))$$

Language to Meaning

at the chair, move forward three steps past the sofa

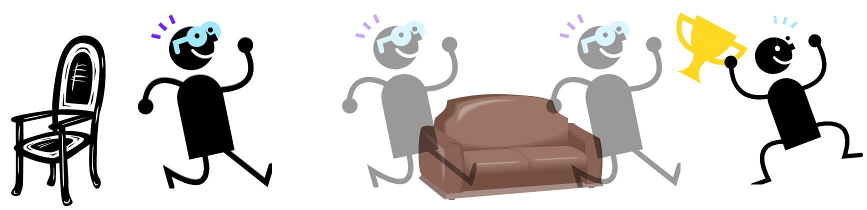
$$\lambda a.pre(a, \iota x.chair(x)) \land move(a) \land len(a, 3) \land dir(a, forward) \land past(a, \iota y.sofa(y))$$

Learn

$$f : \text{sentence} \to \text{logical form}$$

Language to Meaning

at the chair, move forward three steps past the sofa



Learn

$$f : \text{sentence} \rightarrow \text{logical form}$$

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]
LWFG [Muresan 2011]

Learning

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

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

Semantic Modeling

- What logical language to use?
- How to model meaning?

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

Today

Parsing	Combinatory Categorial Grammars
Learning	Unified learning algorithm
Modeling	Best practices for semantics design

Parsing

Learning

Modeling

Parsing | Learning | Modeling

- Lambda calculus
- Parsing with Combinatory Categorial Grammars
- Linear CCGs
- Factored lexicons

Online

Parsing | **Learning** | Modeling

- Structured perceptron
- A unified learning algorithm
- Supervised learning
- Weak supervision

Online

Parsing | Learning | **Modeling**

- Semantic modeling for:
  - Querying databases
  - Referring to physical objects
  - Executing instructions

Online

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]

Parsing Learning Modeling

- Lambda calculus
- Parsing with Combinatory Categorial Grammars
- Linear CCGs
- Factored lexicons

Online

Lambda Calculus

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

$$\lambda a.move(a) \land dir(a, LEFT) \land to(a, \iota y.chair(y)) \land$$
$$pass(a, \mathcal{A}y.sofa(y) \land intersect(\mathcal{A}z.intersection(z), y))$$

[Church 1932]

Lambda Calculus

Base Cases

- Logical constant
- Variable
- Literal
- Lambda term

Lambda Calculus
Logical Constants

- Represent objects in the world

$$NYC, CA, RAINIER, LEFT, \ldots$$
$$located\_in, depart\_date, \ldots$$

Lambda Calculus

Variables

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

$$x, y, z, \ldots$$

Lambda Calculus

Literals

- Represent function application

$$city(AUSTIN)$$

$$located\_in(AUSTIN, TEXAS)$$

Lambda Calculus
Literals

- Represent function application

$$city(AUSTIN)$$

$$located\_in(AUSTIN, TEXAS)$$

Predicate: Logical expression

Arguments: List of logical expressions

Lambda Calculus

Lambda Terms

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

$$\lambda x.city(x)$$

$$\lambda x.\lambda y.located\_in(x, y)$$

Lambda Calculus

Lambda Terms

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

$$\lambda x.city(x)$$

$$\lambda x.\lambda y.located\_in(x, y)$$

Lambda operator

Variable

Body

Lambda Calculus
Quantifiers?

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

$$\forall(\lambda x.big(x) \land apple(x))$$
$$\neg(\exists(\lambda x.lovely(x))$$
$$\iota(\lambda x.beautiful(x) \land grammar(x))$$

Lambda Calculus

Syntactic Sugar

$$\wedge\ (A, \wedge(B, C)) \Leftrightarrow A \wedge B \wedge C$$

$$\vee\ (A, \vee(B, C)) \Leftrightarrow A \vee B \vee C$$

$$\neg(A) \Leftrightarrow \neg A$$

$$\mathcal{Q}(\lambda x.f(x)) \Leftrightarrow \mathcal{Q}x.f(x)$$

$$\text{for } \mathcal{Q} \in \{\iota, \mathcal{A}, \exists, \forall\}$$

$$\lambda x.flight(x) \wedge to(x, move)$$

$$\lambda x.flight(x) \wedge to(x, NYC)$$

$$\lambda x.NYC(x) \wedge x(to, move)$$

✗ $\lambda x.flight(x) \wedge to(x, move)$

✓ $\lambda x.flight(x) \wedge to(x, NYC)$

✗ $\lambda x.NYC(x) \wedge x(to, move)$

Simply Typed Lambda Calculus

- Like lambda calculus
- But, typed

✘ $\lambda x.flight(x) \wedge to(x, move)$

✔ $\lambda x.flight(x) \wedge to(x, NYC)$

✘ $\lambda x.NYC(x) \wedge x(to, move)$

[Church 1940]

Lambda Calculus

Typing

t Truth-value

e Entity

- Simple types
- Complex types

$$< e, t >$$

$$<< e, t >, e >$$

Lambda Calculus

Typing

- Simple types
- Complex types

t Truth-value

e Entity

Type constructor

$$< e, t >$$

$$<< e, t >, e >$$

Domain

Range

Lambda Calculus

Typing

- Simple types
- Complex types

Type constructor

$$< e, t >$$

$$<< e, t >, e >$$

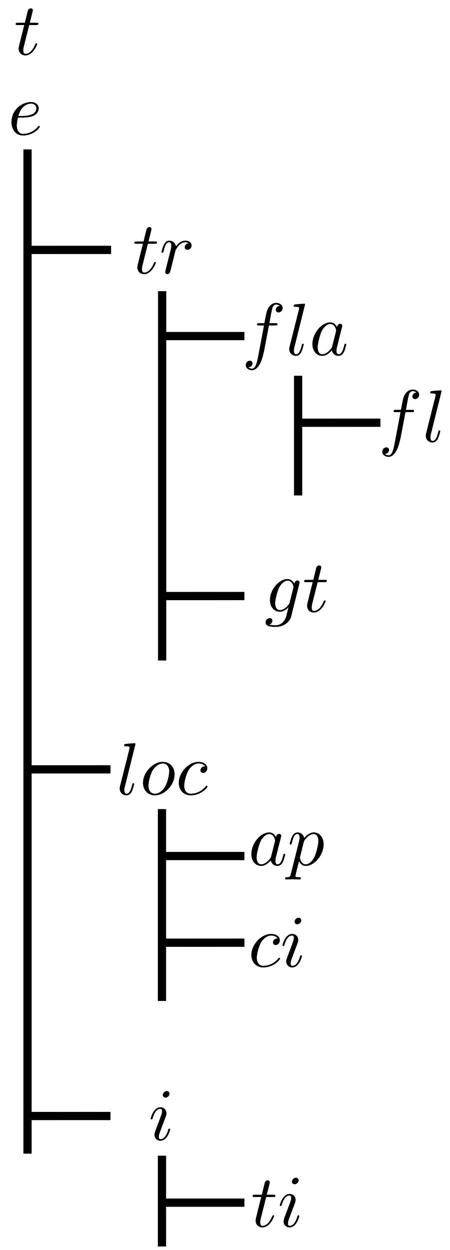
Domain

Range

- Hierarchical typing system

Lambda Calculus

Typing



- Simple types
- Complex types

Type constructor

$$< e, t >$$

$$<< e, t >, e >$$

Domain

Range

- Hierarchical typing system

Simply Typed Lambda Calculus

$$\lambda a.move(a) \land dir(a, LEFT) \land to(a, \iota y.chair(y)) \land$$
$$pass(a, \mathcal{A}y.sofa(y) \land intersect(\mathcal{A}z.intersection(z), y))$$

Type information usually omitted

Capturing Meaning with Lambda Calculus

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4
AZ	Phoenix	2.7

Border

State1	State2
WA	OR
WA	ID
CA	OR
CA	NV
CA	AZ

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA
Wrangel	AK
Si	
Bo	

Show me mountains in states bordering Texas

[Zettlemoyer and Collins 2005]

Capturing Meaning with Lambda Calculus

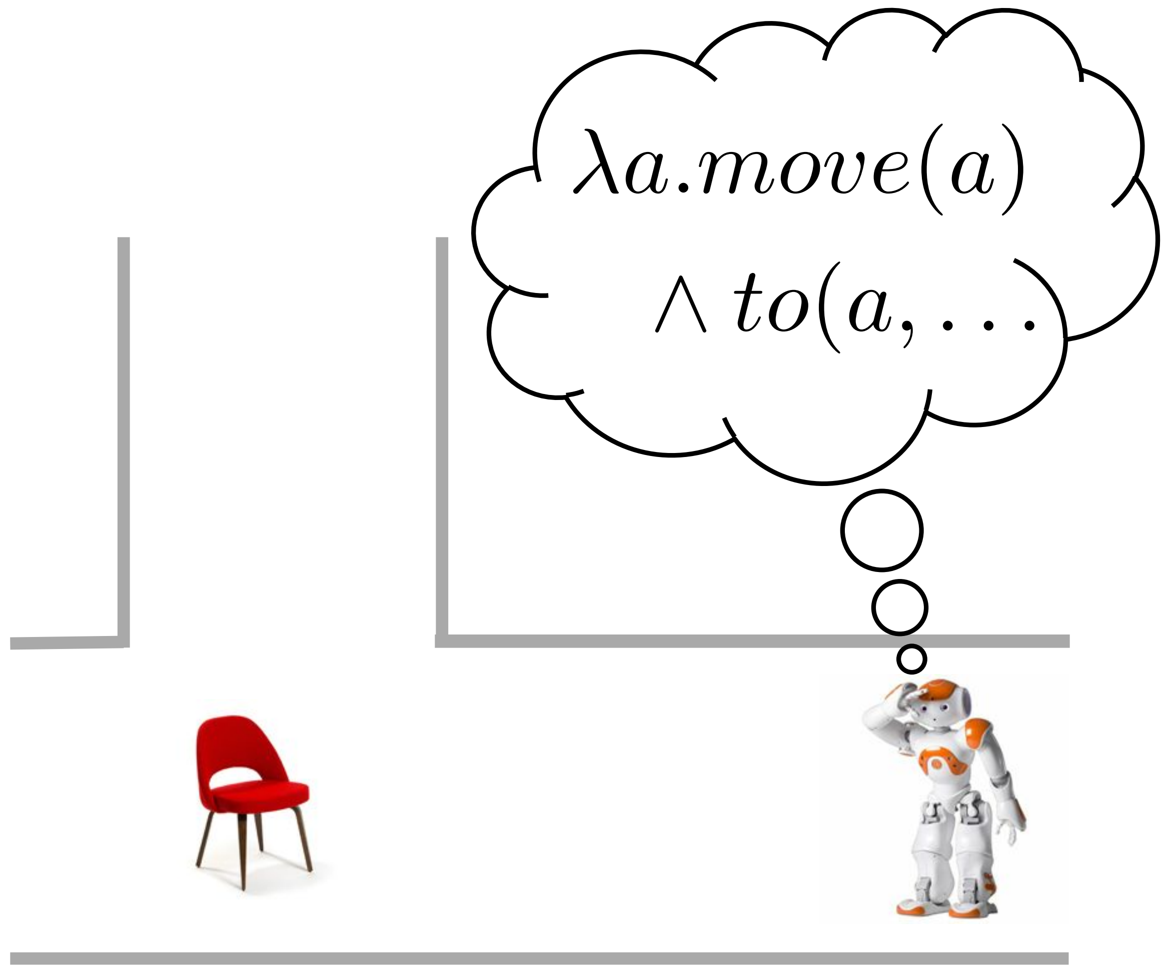
SYSTEM	how can I help you ?
USER	i ‘ d like to fly to new york
SYSTEM	flying to new york . leaving what city ?
USER	from boston on june seven with american airlines
SYSTEM	flying to new york . what date would you like to depart boston ?
USER	june seventh
SYSTEM	do you have a preferred airline ?
USER	american airlines
SYSTEM	o . k . leaving boston to new york on june seventh flying with american airlines . where would you like to go to next ?
USER	back to boston on june tenth

[CONVERSATION CONTINUES]

[Artzi and Zettlemoyer 2011]

Capturing Meaning with Lambda Calculus

go to the chair
and turn right



[Artzi and Zettlemoyer 2013b]

Capturing Meaning with Lambda Calculus

- Flexible representation
- Can capture full complexity of natural language

More on modeling meaning online and later today

Constructing Lambda Calculus Expressions

at the chair, move forward three steps past the sofa

$$\lambda a.pre(a, \iota x.chair(x)) \wedge move(a) \wedge len(a, 3) \wedge$$
$$dir(a, forward) \wedge past(a, \iota y.sofa(y))$$

Combinatory Categorial Grammars

$$
\begin{array}{ccc}
\text{CCG} & \text{is} & \text{fun} \\
\hline
NP & S\backslash NP/ADJ & ADJ \\
CCG & \lambda f.\lambda x.f(x) & \lambda x.fun(x)
\end{array}
$$

$$
\frac{S\backslash NP/ADJ \quad ADJ}{\begin{array}{c} S\backslash NP \\ \lambda x.fun(x) \end{array}} >
$$

$$
\frac{NP \quad S\backslash NP}{\begin{array}{c} S \\ fun(CCG) \end{array}} <
$$

[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

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

- Basic building block
- Capture syntactic and semantic information jointly

CCG Categories

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

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

Semantics

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

$$NP : CCG$$

- λ-calculus expression
- Syntactic type maps to semantic type

CCG Lexical Entries

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

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

CCG Lexical Entries

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

Natural Language

CCG Category

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

CCG Lexicons

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

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

$$\text{CCG} \vdash NP : CCG$$

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

Between CCGs and CFGs

	CFGs	CCGs
Combination operations	Many	Few
Parse tree nodes	Non-terminals	Categories
Syntactic symbols	Few dozen	Handful, but can combine
Paired with words	POS tags	Categories

Parsing with CCGs

$$\frac{\text{CCG}}{\begin{array}{c}NP\\ CCG\end{array}} \quad \frac{\text{is}}{\begin{array}{c}S\backslash NP/ADJ\\ \lambda f.\lambda x.f(x)\end{array}} \quad \frac{\text{fun}}{\begin{array}{c}ADJ\\ \lambda x.fun(x)\end{array}}$$

Use lexicon to match words and phrases with their categories

CCG Operations

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

CCG Operations
Application

$$B : g \quad A\backslash B : f \Rightarrow A : f(g) \quad (<)$$

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

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

CCG Operations

Application

Argument | Function | Result

$$B : g \quad A\backslash B : f \Rightarrow A : f(g) \qquad (<)$$

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

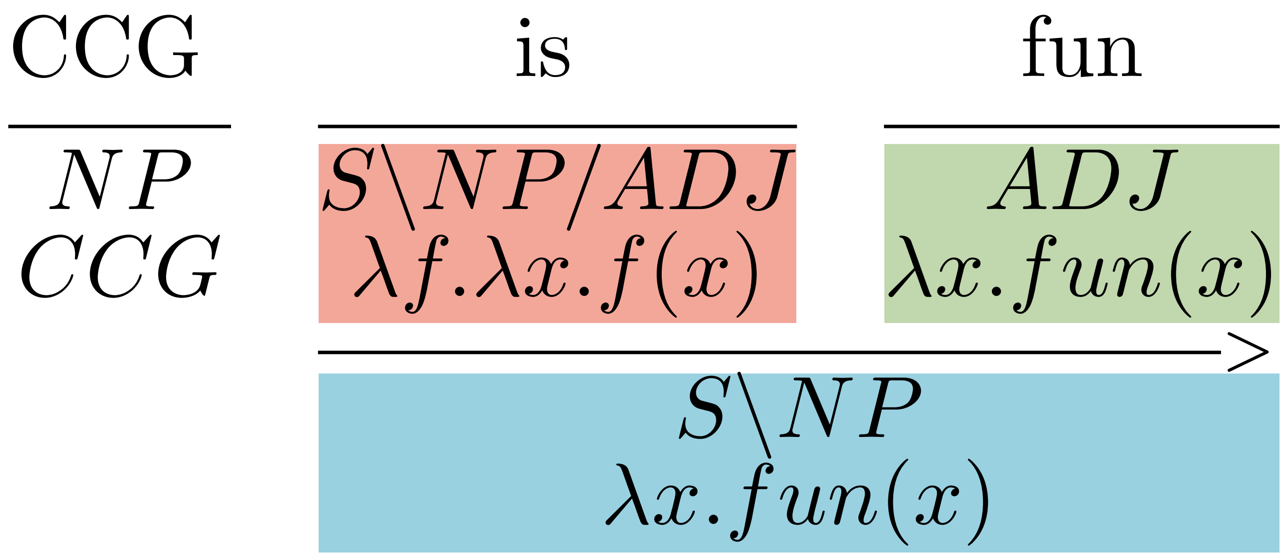
- Equivalent to function application
- Two directions: forward and backward
  - Determined by slash direction

Parsing with CCGs

$$\frac{\text{CCG}}{\begin{array}{c}NP\\ CCG\end{array}} \quad \frac{\text{is}}{\begin{array}{c}S\backslash NP/ADJ\\ \lambda f.\lambda x.f(x)\end{array}} \quad \frac{\text{fun}}{\begin{array}{c}ADJ\\ \lambda x.fun(x)\end{array}}$$

Use lexicon to match words and phrases with their categories

Parsing with CCGs



Combine categories using operators

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

Parsing with CCGs

CCG	is	fun
NP	$S\backslash NP/ADJ$	ADJ
CCG	$\lambda f.\lambda x.f(x)$	$\lambda x.fun(x)$

$$\text{is fun} \; (>) : \quad S\backslash NP \quad \lambda x.fun(x)$$

$$\text{CCG is fun} \; (<) : \quad S \quad fun(CCG)$$

Combine categories using operators

$$B : g \quad A\backslash B : f \Rightarrow A : f(g) \quad (<)$$

Parsing with CCGs

Composed adjectives

square blue or round yellow pillow

Non-standard coordination

CCG Operations
Composition

$$A/B : f \quad B/C : g \Rightarrow A/C : \lambda x.f(g(x)) \quad (> B)$$

$$B\backslash C : g \quad A\backslash B : f \Rightarrow A\backslash C : \lambda x.f(g(x)) \quad (< B)$$

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

* Formal definition of logical composition in supplementary slides

CCG Operations

Composition

f g $f \circ g$

$$A/B : f \quad B/C : g \Rightarrow A/C : \lambda x.f(g(x)) \qquad (> B)$$

$$B\backslash C : g \quad A\backslash B : f \Rightarrow A\backslash C : \lambda x.f(g(x)) \qquad (< B)$$

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

* Formal definition of logical composition in supplementary slides

CCG Operations
Type Shifting

$$ADJ : \lambda x.g(x) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)$$

$$PP : \lambda x.g(x) \Rightarrow N\backslash N : \lambda f.\lambda x.f(x) \wedge g(x)$$

$$AP : \lambda e.g(e) \Rightarrow S\backslash S : \lambda f.\lambda e.f(e) \wedge g(e)$$

$$AP : \lambda e.g(e) \Rightarrow S/S : \lambda f.\lambda e.f(e) \wedge g(e)$$

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

CCG Operations

Type Shifting

Input		Output
$ADJ : \lambda x.g(x)$	$\Rightarrow$	$N/N : \lambda f.\lambda x.f(x) \wedge g(x)$

$$PP : \lambda x.g(x) \Rightarrow N\backslash N : \lambda f.\lambda x.f(x) \wedge g(x)$$

$$AP : \lambda e.g(e) \Rightarrow S\backslash S : \lambda f.\lambda e.f(e) \wedge g(e)$$

$$AP : \lambda e.g(e) \Rightarrow S/S : \lambda f.\lambda e.f(e) \wedge g(e)$$

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

CCG Operations

Type Shifting

Input → Output

$$ADJ : \lambda x.g(x) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)$$

$$PP : \lambda x.g(x) \Rightarrow N\backslash N : \lambda f.\lambda x.f(x) \wedge g(x)$$

$$AP : \lambda e.g(e) \Rightarrow S\backslash S : \lambda f.\lambda e.f(e) \wedge g(e)$$

Topicalization

$$AP : \lambda e.g(e) \Rightarrow S/S : \lambda f.\lambda e.f(e) \wedge g(e)$$

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

CCG Operations
Coordination

$$\text{and} \vdash C : conj$$
$$\text{or} \vdash C : disj$$

- 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

$$\frac{\text{square}}{\begin{array}{c}ADJ\\ \lambda x.square(x)\end{array}} \quad \frac{\text{blue}}{\begin{array}{c}ADJ\\ \lambda x.blue(x)\end{array}} \quad \frac{\text{or}}{\begin{array}{c}C\\ disj\end{array}} \quad \frac{\text{round}}{\begin{array}{c}ADJ\\ \lambda x.round(x)\end{array}} \quad \frac{\text{yellow}}{\begin{array}{c}ADJ\\ \lambda x.yellow(x)\end{array}} \quad \frac{\text{pillow}}{\begin{array}{c}N\\ \lambda x.pillow(x)\end{array}}$$

Use lexicon to match words and phrases with their categories

Parsing with CCGs

square	blue	or	round	yellow	pillow
ADJ $\lambda x.square(x)$	ADJ $\lambda x.blue(x)$	C $disj$	ADJ $\lambda x.round(x)$	ADJ $\lambda x.yellow(x)$	N $\lambda x.pillow(x)$
N/N $\lambda f.\lambda x.f(x) \wedge square(x)$					

Shift adjectives to combine

$$ADJ : \lambda x.g(x) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)$$

Parsing with CCGs

square	blue	or	round	yellow	pillow
ADJ	ADJ	C	ADJ	ADJ	N
$\lambda x.square(x)$	$\lambda x.blue(x)$	$disj$	$\lambda x.round(x)$	$\lambda x.yellow(x)$	$\lambda x.pillow(x)$
N/N	N/N		N/N	N/N	
$\lambda f.\lambda x.f(x) \wedge square(x)$	$\lambda f.\lambda x.f(x) \wedge blue(x)$		$\lambda f.\lambda x.f(x) \wedge round(x)$	$\lambda f.\lambda x.f(x) \wedge yellow(x)$	

Shift adjectives to combine

$$ADJ : \lambda x.g(x) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)$$

Parsing with CCGs

square	blue	or	round	yellow	pillow
ADJ $\lambda x.square(x)$	ADJ $\lambda x.blue(x)$	C $disj$	ADJ $\lambda x.round(x)$	ADJ $\lambda x.yellow(x)$	N $\lambda x.pillow(x)$
N/N $\lambda f.\lambda x.f(x) \wedge square(x)$	N/N $\lambda f.\lambda x.f(x) \wedge blue(x)$		N/N $\lambda f.\lambda x.f(x) \wedge round(x)$	N/N $\lambda f.\lambda x.f(x) \wedge yellow(x)$	
$>$B: N/N $\lambda f.\lambda x.f(x) \wedge square(x) \wedge blue(x)$			$>$B: N/N $\lambda f.\lambda x.f(x) \wedge round(x) \wedge yellow(x)$		

Compose pairs of adjectives

$$A/B : f \quad B/C : g \Rightarrow A/C : \lambda x.f(g(x)) \quad (> B)$$

Parsing with CCGs

square	blue	or	round	yellow	pillow
ADJ $\lambda x.square(x)$	ADJ $\lambda x.blue(x)$	C $disj$	ADJ $\lambda x.round(x)$	ADJ $\lambda x.yellow(x)$	N $\lambda x.pillow(x)$
N/N $\lambda f.\lambda x.f(x) \wedge square(x)$	N/N $\lambda f.\lambda x.f(x) \wedge blue(x)$		N/N $\lambda f.\lambda x.f(x) \wedge round(x)$	N/N $\lambda f.\lambda x.f(x) \wedge yellow(x)$	

square blue $>\mathbf{B}$:

$$N/N \quad \lambda f.\lambda x.f(x) \wedge square(x) \wedge blue(x)$$

round yellow $>\mathbf{B}$:

$$N/N \quad \lambda f.\lambda x.f(x) \wedge round(x) \wedge yellow(x)$$

square blue or round yellow $<\Phi>$:

$$N/N \quad \lambda f.\lambda x.f(x) \wedge ((square(x) \wedge blue(x)) \vee (round(x) \wedge yellow(x)))$$

Coordinate composed adjectives

Parsing with CCGs

square	blue	or	round	yellow	pillow
ADJ $\lambda x.square(x)$	ADJ $\lambda x.blue(x)$	C $disj$	ADJ $\lambda x.round(x)$	ADJ $\lambda x.yellow(x)$	N $\lambda x.pillow(x)$
N/N $\lambda f.\lambda x.f(x) \wedge square(x)$	N/N $\lambda f.\lambda x.f(x) \wedge blue(x)$		N/N $\lambda f.\lambda x.f(x) \wedge round(x)$	N/N $\lambda f.\lambda x.f(x) \wedge yellow(x)$	

$>$**B**: N/N — $\lambda f.\lambda x.f(x) \wedge square(x) \wedge blue(x)$

$>$**B**: N/N — $\lambda f.\lambda x.f(x) \wedge round(x) \wedge yellow(x)$

$<\Phi>$: N/N — $\lambda f.\lambda x.f(x) \wedge ((square(x) \wedge blue(x)) \vee (round(x) \wedge yellow(x)))$

$>$: N — $\lambda x.pillow(x) \wedge ((square(x) \wedge blue(x)) \vee (round(x) \wedge yellow(x)))$

Apply coordinated adjectives to noun

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

Parsing with CCGs

x: CCG is fun

y:

CCG	is	fun
NP	$S\backslash NP/ADJ$	ADJ
CCG	$\lambda f.\lambda x.f(x)$	$\lambda x.fun(x)$

$$\frac{\lambda f.\lambda x.f(x) \quad \lambda x.fun(x)}{S\backslash NP \;\; \lambda x.fun(x)} >$$

$$\frac{CCG \quad S\backslash NP}{S} <$$

z: $fun(CCG)$

Lexical Ambiguity + Many parsing decisions → Many potential trees and LFs

Weighted Linear CCGs

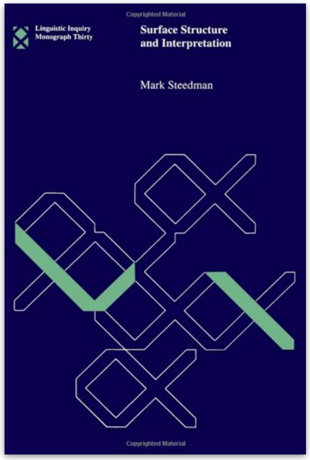
- Given a weighted linear model:
  - CCG lexicon Λ
  - Feature function $f : X \times Y \to \mathbb{R}^m$
  - Weights $w \in \mathbb{R}^m$
- The best parse is:

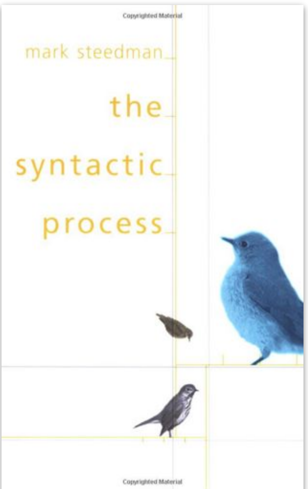
$$y^* = \arg\max_y w \cdot f(x, y)$$

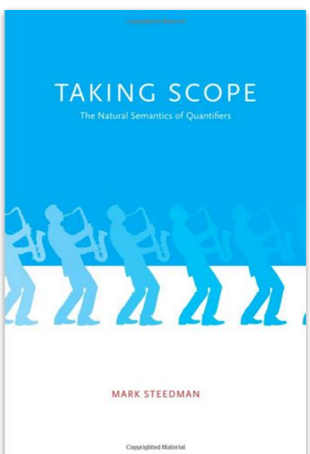
- We consider all possible parses y for sentence x given the lexicon Λ

Parsing Algorithms

- Syntax-only CCG parsing has polynomial time CKY-style algorithms
- Parsing with semantics requires entire category as chart signature
  - e.g., $ADJ : \lambda x.fun(x)$
- In practice, prune to top-N for each span
  - Approximate, but polynomial time







More on CCGs

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

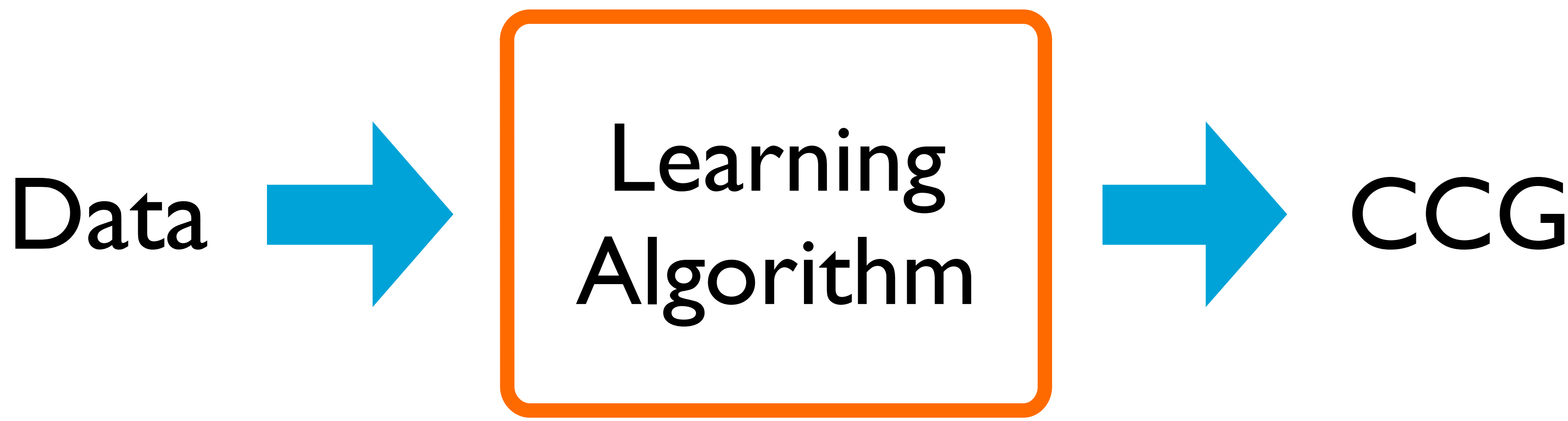
[Steedman 1996; 2000; 2011; Granroth and Steedman 2012]

Parsing | Learning | Modeling

- Lambda calculus
- Parsing with Combinatory Categorial Grammars
- Linear CCGs
- Factored lexicons

Online

Learning



- What kind of data/supervision we can use?
- What do we need to learn?

Parsing as Structure Prediction

show me	flights	to	Boston
S/N	N	PP/NP	NP
$\lambda f.f$	$\lambda x.flight(x)$	$\lambda y.\lambda x.to(x,y)$	$BOSTON$

$$\frac{PP/NP \quad NP}{PP} > \qquad \lambda x.to(x, BOSTON)$$

$$N\backslash N \qquad \lambda f.\lambda x.f(x) \wedge to(x, BOSTON)$$

$$\frac{N \quad N\backslash N}{N} < \qquad \lambda x.flight(x) \wedge to(x, BOSTON)$$

$$\frac{S/N \quad N}{S} > \qquad \lambda x.flight(x) \wedge to(x, BOSTON)$$

Learning CCG

show me | flights | to | Boston

S/N — $\lambda f.f$

N — $\lambda x.flight(x)$

PP/NP — $\lambda y.\lambda x.to(x,y)$

NP — $BOSTON$

$>$ PP — $\lambda x.to(x, BOSTON)$

$N \backslash N$ — $\lambda f.\lambda x.f(x) \wedge to(x, BOSTON)$

$<$ N — $\lambda x.flight(x) \wedge to(x, BOSTON)$

$>$ S — $\lambda x.flight(x) \wedge to(x, BOSTON)$

Lexicon

Combinators

Predefined

w

Supervised Data

show me	flights	to	Boston
S/N	N	PP/NP	NP
$\lambda f.f$	$\lambda x.flight(x)$	$\lambda y.\lambda x.to(x,y)$	$BOSTON$

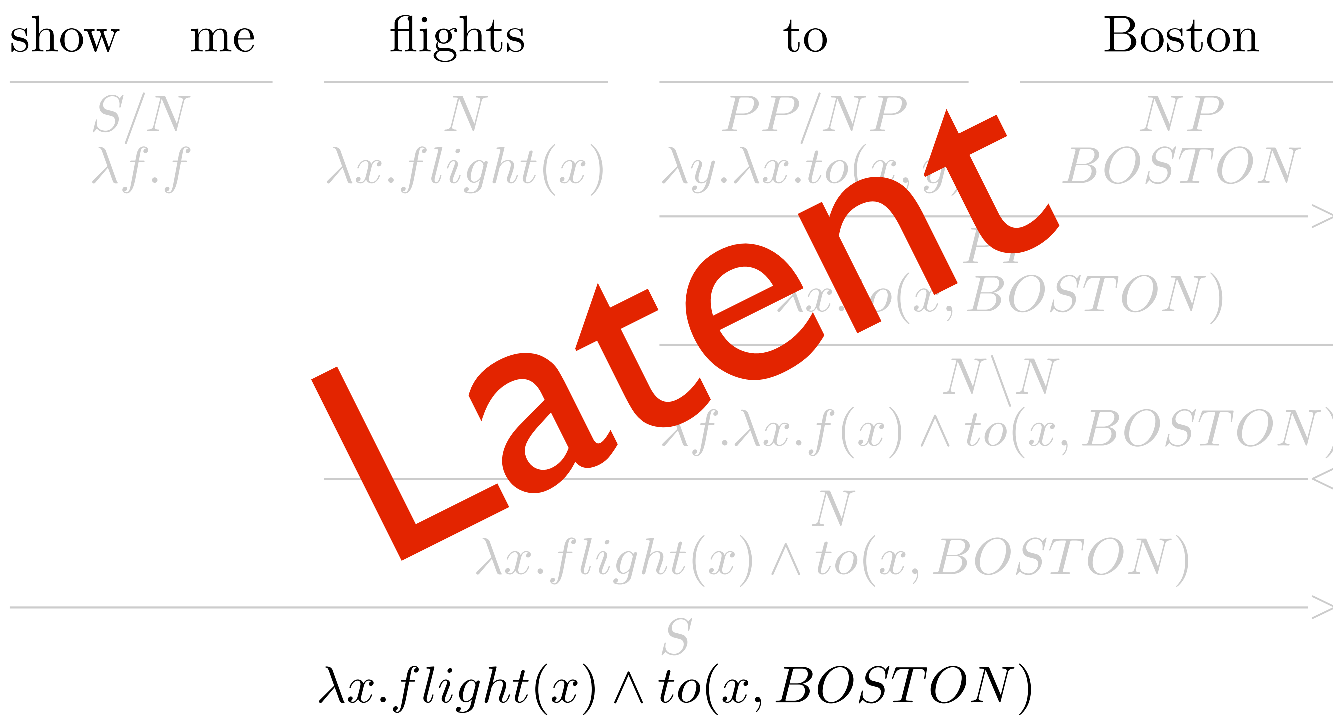
$$\frac{PP/NP \quad \lambda y.\lambda x.to(x,y) \qquad NP \quad BOSTON}{PP \quad \lambda x.to(x,BOSTON)} >$$

$$\frac{PP \quad \lambda x.to(x,BOSTON)}{N\backslash N \quad \lambda f.\lambda x.f(x) \land to(x,BOSTON)}$$

$$\frac{N \quad \lambda x.flight(x) \qquad N\backslash N \quad \lambda f.\lambda x.f(x) \land to(x,BOSTON)}{N \quad \lambda x.flight(x) \land to(x,BOSTON)} <$$

$$\frac{S/N \quad \lambda f.f \qquad N \quad \lambda x.flight(x) \land to(x,BOSTON)}{S \quad \lambda x.flight(x) \land to(x,BOSTON)} >$$

Supervised Data



Supervised Data

Supervised learning is done from pairs of sentences and logical forms

Show me flights to Boston

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

I need a flight from baltimore to seattle

$\lambda x.flight(x) \wedge from(x, BALTIMORE) \wedge to(x, SEATTLE)$

what ground transportation is available in san francisco

$\lambda x.ground\_transport(x) \wedge to\_city(x, SF)$

[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?

$$New\ Mexico$$

$$argmax(\lambda x.state(x) \\ \wedge border(x, TX), \lambda y.size(y))$$

$$argmax(\lambda x.river(x) \\ \wedge in(x, TX), \lambda y.size(y))$$

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

Weak Supervision

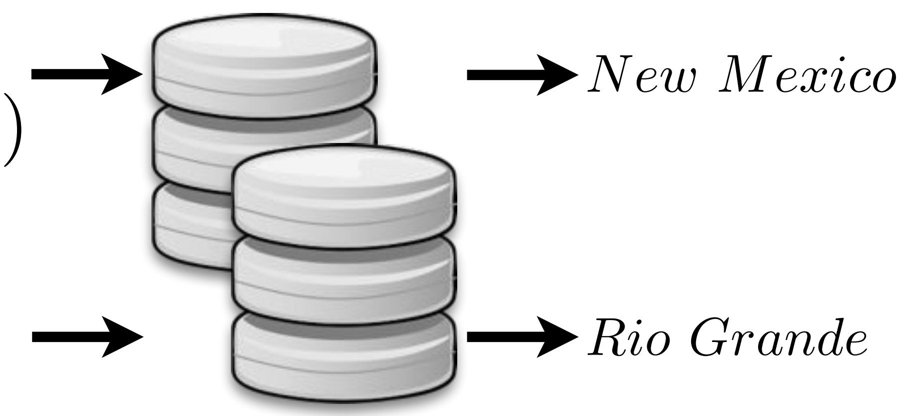
Learning from Query Answers

What is the largest state that borders Texas?

$$New\ Mexico$$

$$argmax(\lambda x.state(x) \wedge border(x, TX), \lambda y.size(y))$$

$$argmax(\lambda x.river(x) \wedge in(x, TX), \lambda y.size(y))$$



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

Weak Supervision

Learning from Query Answers

What is the largest state that borders Texas?

$$New\ Mexico$$

$$argmax(\lambda x.state(x) \wedge border(x, TX), \lambda y.size(y)) \rightarrow New\ Mexico \checkmark$$

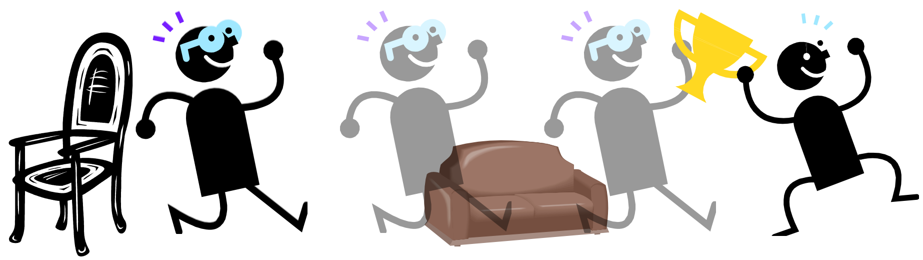
$$argmax(\lambda x.river(x) \wedge in(x, TX), \lambda y.size(y)) \rightarrow Rio\ Grande \times$$

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

Weak Supervision

Learning from Demonstrations

at the chair, move forward three steps past the sofa

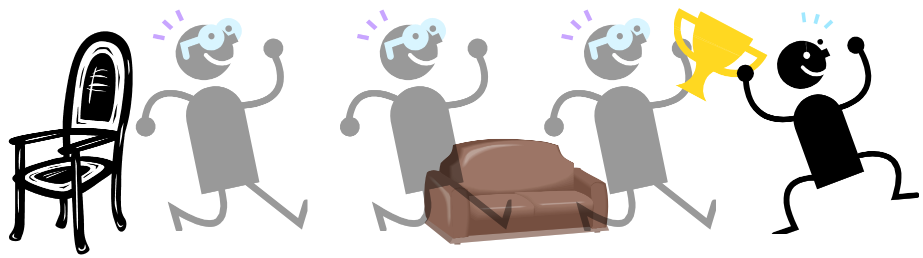


[Chen and Mooney 2011; Kim and Mooney 2012; Artzi and Zettlemoyer 2013b]

Weak Supervision

Learning from Demonstrations

at the chair, move forward three steps past the sofa



Some examples from other domains:

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

[Chen and Mooney 2011; Kim and Mooney 2012; Artzi and Zettlemoyer 2013b]

Weak Supervision
Learning from Conversation Logs

SYSTEM how can I help you ? (OPEN_TASK)

USER i ‘ d like to fly to new york

SYSTEM flying to new york . (CONFIRM: $from(fl, ATL)$) leaving what city ? (ASK: $\lambda x.from(fl,x)$)

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: $\lambda x.date(fl,x) \wedge to(fl, BOS)$)

USER june seventh

[CONVERSATION CONTINUES]

[Artzi and Zettlemoyer 2011]

Parsing | **Learning** | Modeling

- Structured perceptron
- A unified learning algorithm
- Supervised learning
- Weak supervision — Online

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

- Simple additive updates
  - Only requires efficient decoding (argmax)
  - Closely related to MaxEn
    feature rich models
  - Provably finds linear separ
    updates, if one exists
- Challenge: learning with hic

Derivations in the complete tutorial

Hidden Variable Perceptron

- No known convergence guarantees
  - Log-linear version is non-convex
- Simple and easy to implement
  - Works well with careful initialization
- Modifications for semantic parsing
  - Lots of different hidden information
  - Can add a margin constraint, do probabilistic version, etc.

Unified Learning Algorithm

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

Learning Choices

Validation Function

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

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

Lexical Generation Procedure

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

- Given:
  - sentence x
  - validation function $\mathcal{V}$
  - lexicon Λ
  - parameters θ
- Produce a overly general set of lexical entries

Unified Learning Algorithm

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

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

Step 1: (Lexical generation)

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

- Online
- 2 steps:
  - Lexical generation
  - Parameter update

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

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

Step 1: (Lexical generation)

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Initialize parameters and lexicon

θ weights

Λ_0 initial lexicon

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

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

Step 1: (Lexical generation)

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Iterate over data

T # iterations

n # samples

Initialize θ using Λ_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 θ and lexicon Λ

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

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

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 θ and lexicon Λ

Generate a large set of potential lexical entries

θ weights

x sentence

$\mathcal{V}$ validation function

$GENLEX(x, \mathcal{V}; \lambda, \theta)$
lexical generation function

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

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

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 θ and lexicon Λ

Generate a large set of potential lexical entries

θ weights

x sentence

$\mathcal{V}$ validation function

$GENLEX(x, \mathcal{V}; \lambda, \theta)$
lexical generation function

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

$\mathcal{Y}$ all parses

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

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

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 θ and lexicon Λ

Generate a large set of potential lexical entries

θ weights

x sentence

$\mathcal{V}$ validation function

$GENLEX(x, \mathcal{V}; \lambda, \theta)$
lexical generation function

Procedure to propose potential new lexical entries for a sentence

Initialize θ using Λ_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 θ and lexicon Λ

Get top parses

x sentence

k beam size

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

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

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

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 θ and lexicon Λ

Get lexical entries from highest scoring valid parses

θ weights

$\mathcal{V}$ validation function

$LEX(y)$ set of lexical entries

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

$MAXV_i(Y;\theta) =$
$\{y | \forall y' \in Y, \langle \theta, \Phi_i(y') \rangle \leq \langle \theta, \Phi_i(y) \rangle \wedge \mathcal{V}_i(y)\}$

Initialize θ using Λ_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: $\Lambda \leftarrow \Lambda \cup \lambda_i$

Update model's lexicon

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Initialize θ using Λ_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 $B_i \leftarrow \{e | e \in GEN(x_i; \Lambda) \wedge \neg \mathcal{V}_i(y)\}$

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

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

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

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 θ and lexicon Λ

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

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

Step 1: (Lexical generation)

Step 2: (Update parameters)

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

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

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

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

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 θ and lexicon Λ

Re-parse and group all parses into 'good' and 'bad' sets

θ weights

x sentence

$\mathcal{V}$ validation function

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

$$MAXV_i(Y; \theta) = \{y | \forall y' \in Y, \langle \theta, \Phi_i(y') \rangle \leq \langle \theta, \Phi_i(y) \rangle \wedge \mathcal{V}_i(y) = 1\}$$

Initialize θ using Λ_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 $B_i \leftarrow \{e | e \in GEN(x_i; \Lambda) \wedge \neg \mathcal{V}_i(y)\}$

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

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 θ and lexicon Λ

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

θ weights

γ margin

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

$\Delta_i(y, y') = |\Phi_i(y) - \Phi_i(y')|_1$

Initialize θ using Λ_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 $B_i \leftarrow \{e | e \in GEN(x_i; \Lambda) \wedge \neg \mathcal{V}_i(y)\}$

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

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

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

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 θ and lexicon Λ

Update towards violating 'good' parses and against violating 'bad' parses

θ weights

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

Initialize θ using Λ_0 , $\Lambda \leftarrow \Lambda_0$

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

Step 1: (Lexical generation)

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

Return grammar

θ weights

Λ lexicon

Features and Initialization

Feature Classes

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

Lexicon Initialization

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

Initial Weights

- Positive weight for initial lexical indicator features

Unified Learning Algorithm Extensions

- Loss-sensitive learning
  - Applied to learning from conversations
- Stochastic gradient descent
  - Approximate expectation computation

[Artzi and Zettlemoyer 2011; Zettlemoyer and Collins 2005]

Unified Learning Algorithm

$$\mathcal{V} \text{ validation function}$$

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

$$\text{lexical generation function}$$

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

Unified Learning Algorithm

Supervised

$\mathcal{V}$

Template-based $GENLEX$

Unification-based $GENLEX$

Weakly Supervised

$\mathcal{V}$

Template-based $GENLEX$

Supervised Learning

show me the afternoon flights from LA to boston

$$\lambda x.flight(x) \wedge during(x, AFTERNOON) \wedge from(x, LA) \wedge to(x, BOS)$$

Supervised Learning

show me the afternoon flights from LA to boston

$$\lambda x.flight(x) \wedge during(x, AFTERNOON) \wedge from(x, LA) \wedge to(x, BOS)$$

Parse structure is latent

Supervised Validation Function

- Validate logical form against gold label

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

y parse

z_i labeled logical form

$LF(y)$ logical form at the root of y

Supervised Template-based

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

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

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

Supervised Template-based GENLEX

- Use templates to constrain lexical entries structure
- For example: from a small annotated dataset

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

[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

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

I want a flight to new york

$$\lambda x.flight(x) \wedge to(x, NYC)$$

All possible sub-strings

I want

a flight

flight

flight to new

$\ldots$

Supervised Template-based

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

I want a flight to new york

$\lambda x.flight(x) \wedge to(x, NYC)$

All logical constants from labeled logical form

I want

a flight

flight

flight to new

$\ldots$

$flight$

to

NYC

Supervised Template-based

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

I want a flight to new york

$\lambda x.flight(x) \wedge to(x, NYC)$

I want
a flight
flight
flight to new
$\ldots$

×

$flight$
to
NYC

⇒

Create lexemes

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

Supervised Template-based

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

I want a flight to new york

$$\lambda x.flight(x) \wedge to(x, NYC)$$

I want
a flight
flight
flight to new
$\ldots$

×

$flight$
to
NYC

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

Initialize templates

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

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

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

Fast Parsing with Pruning

I want a flight from New York to Boston on Delta

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

$$\ldots \quad \frac{\text{form}}{\begin{array}{c} PP/NP \\ \lambda x.\lambda y.to(y,x) \end{array}} \quad \frac{\text{New York}}{\begin{array}{c} NP \\ NYC \end{array}} \quad \frac{\text{to}}{\begin{array}{c} PP/NP \\ \lambda x.\lambda y.to(y,x) \end{array}} \quad \frac{\text{Boston}}{\begin{array}{c} NP \\ BOS \end{array}} \quad \ldots$$

Fast Parsing with Pruning

I want a flight from New York to Boston on Delta

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

$$\ldots \quad \begin{array}{cc} \text{form} & \text{New York} \\ \hline PP/NP & NP \\ \lambda x.\lambda y.to(y,x) & NYC \\ \hline \multicolumn{2}{c}{PP} \\ \multicolumn{2}{c}{\lambda y.to(y, NYC)} \end{array} {>} \quad \begin{array}{cc} \text{to} & \text{Boston} \\ \hline PP/NP & NP \\ \lambda x.\lambda y.to(y,x) & BOS \\ \hline \multicolumn{2}{c}{PP} \\ \multicolumn{2}{c}{\lambda y.to(y, BOS)} \end{array} {>} \quad \ldots$$

Fast Parsing with Pruning

I want a flight from New York to Boston on Delta

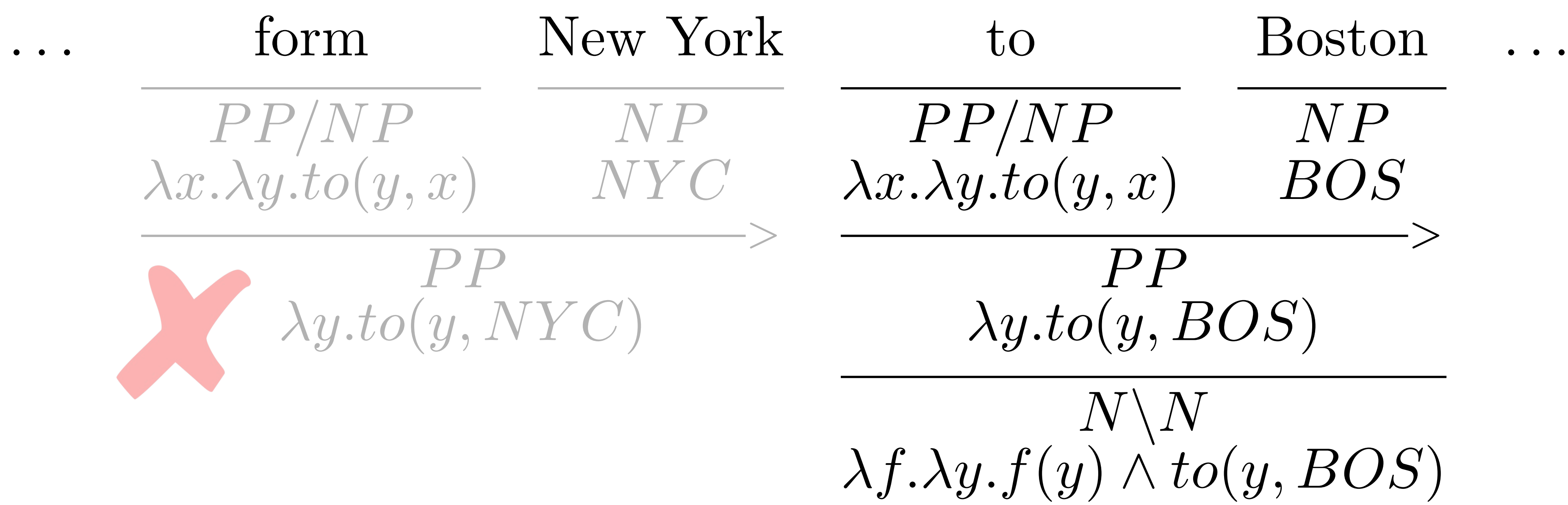
$$\lambda x.from(x, NYC) \wedge to(x, BOS) \wedge carrier(x, DL)$$

$$\ldots \quad \dfrac{\dfrac{\text{form}}{\begin{array}{c} PP/NP \\ \lambda x.\lambda y.to(y,x) \end{array}} \quad \dfrac{\text{New York}}{\begin{array}{c} NP \\ NYC \end{array}}}{\begin{array}{c} PP \\ \lambda y.to(y, NYC) \end{array}} > \quad \dfrac{\dfrac{\text{to}}{\begin{array}{c} PP/NP \\ \lambda x.\lambda y.to(y,x) \end{array}} \quad \dfrac{\text{Boston}}{\begin{array}{c} NP \\ BOS \end{array}}}{\begin{array}{c} PP \\ \lambda y.to(y, BOS) \end{array}} > \quad \ldots$$

Fast Parsing with Pruning

I want a flight from New York to Boston on Delta

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



Supervised Template-based GENLEX

Summary

No initial expert knowledge	
Creates compact lexicons	✓
Language independent	
Representation independent	
Easily inject linguistic knowledge	✓
Weakly supervised learning	✓

Unification-based GENLEX

- Automatically learns the templates
  - Can be applied to any language and many different approaches for semantic modeling
- Two step process
  - Initialize lexicon with labeled logical forms
  - "Reverse" parsing operations to split lexical entries

[Kwiatkowski et al. 2010]

Unification-based GENLEX

- Initialize lexicon with labeled logical forms

For every labeled training example:

I want a flight to Boston

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

Initialize the lexicon with:

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

Unification-based GENLEX

- Splitting lexical entries

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

⬇

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

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

Unification-based GENLEX

- Splitting lexical entries

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

⇓

$$\text{I want a flight} \vdash S/(S|NP) : \lambda f.\lambda x.flight(x) \wedge f(x)$$
$$\text{to Boston} \vdash S|NP : \lambda x.to(x, BOS)$$

Many possible phrase pairs × Many possible category pairs

Unification-based GENLEX

- Splitting lexical entries

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

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

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

Many possible phrase pairs × Man cat

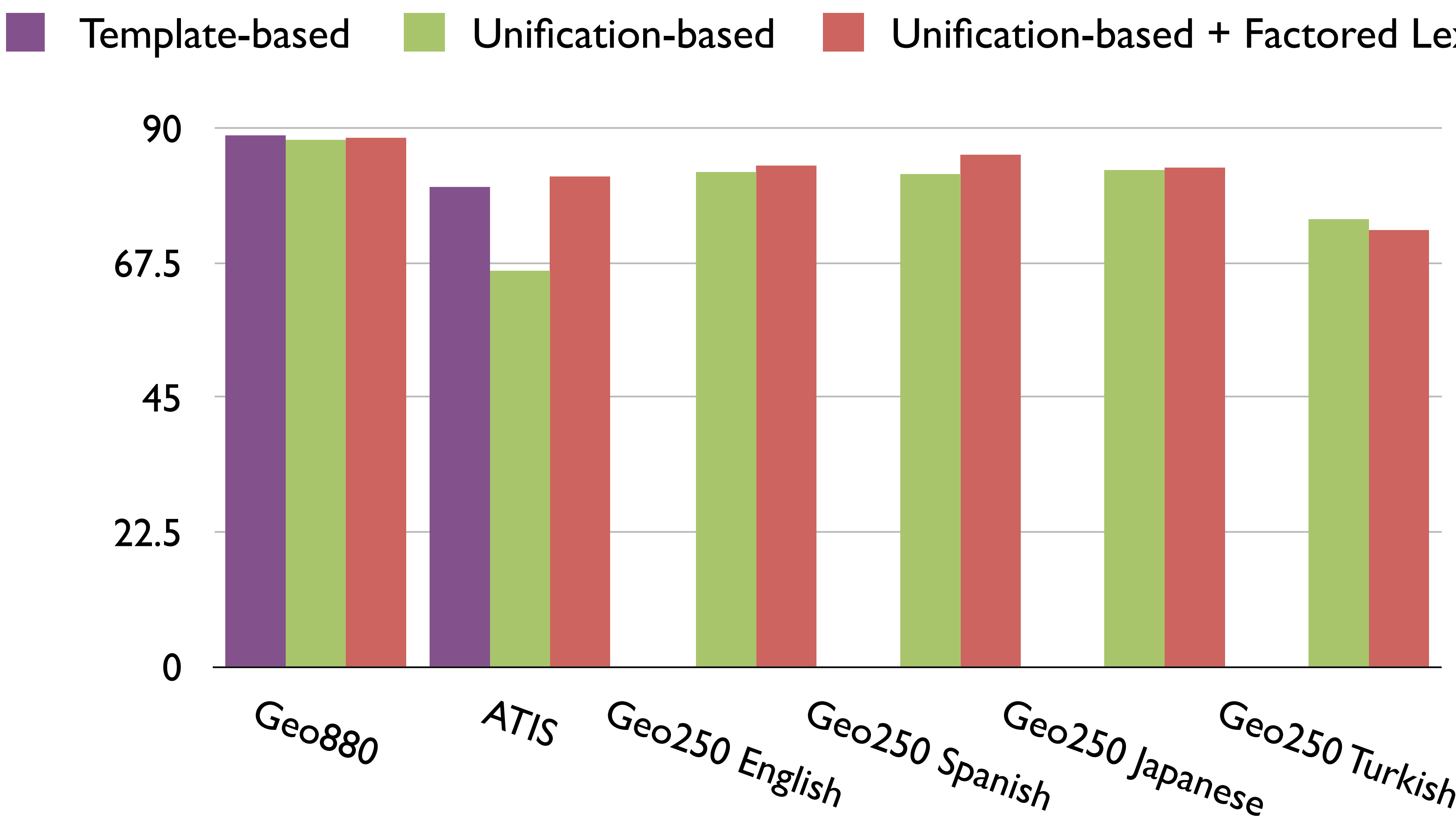
More details in the complete tutorial

Experiments

- Two database corpora:
  - Geo880/Geo250 [Zelle and Mooney 1996; Tang and Mooney 2001]
  - ATIS [Dahl et al. 1994]
- Learning from sentences paired with logical forms
- Comparing template-based and unification-based GENLEX methods

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

Results



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

GENLEX Comparison

	Templates	Unification
No initial expert knowledge		✓
Creates compact lexicons	✓	
Language independent		✓
Representation independent		✓
Easily inject linguistic knowledge	✓	
Weakly supervised learning	✓	

GENLEX Comparison

	Templates	Unification
No initial expert knowledge		✓
Creates compact lexicons	✓	
Language independent		✓
Representation independent		✓
Easily inject linguistic knowledge	✓	
Weakly supervised learning	✓	?

Parsing | **Learning** | Modeling

- Structured perceptron
- A unified learning algorithm
- Supervised learning
- Weak supervision — Online

Modeling

Show me all papers about semantic parsing

↓ Parsing with CCG

$$\lambda x.paper(x) \wedge topic(x, SEMPAR)$$

Modeling

Show me all papers about semantic parsing

Parsing with CCG

$$\lambda x.paper(x) \wedge topic(x, SEMPAR)$$

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

Parsing

Learning

Modeling

- Semantic modeling for:
  - Querying databases
  - Referring to physical objects
  - Executing instructions

Online

Querying Databases

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4
AZ	Phoenix	2.7
WA	Olympia	4.1
NY	Albany	17.5
IL	Springfield	11.4

Border

State1	State2
WA	OR
WA	ID
CA	OR
CA	NV
CA	AZ

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA
Wrangel	AK
Sill	CA
Bo[illegible]	[illegible]
Elbe[illegible]	[illegible]

[Zettlemoyer and Collins 2005]

Querying Databases

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4
AZ	Phoenix	2.7

Border

State1	State2
WA	OR
WA	ID
CA	OR
CA	NV

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

What is the capital of Arizona?

How many states border California?

What is the largest state?

Querying Databases

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4
AZ	Phoenix	2.7

Border

State1	State2
WA	OR
WA	ID
CA	OR
CA	NV

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

What is the capital of Arizona?

How many states border California?

What is the largest state?

Noun Phrases

Querying Databases

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4
AZ	Phoenix	2.7

Border

State1	State2
WA	OR
WA	ID
CA	OR
CA	NV

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

What is the capital of Arizona?

How many states border California?

What is the largest state?

Verbs

Querying Databases

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4
AZ	Phoenix	2.7

Border

State1	State2
WA	OR
WA	ID
CA	OR
CA	NV

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

What is the capital of Arizona?

How many states border California?

What is the largest state?

Nouns

Querying Databases

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4
AZ	Phoenix	2.7

Border

State1	State2
WA	OR
WA	ID
CA	OR
CA	NV

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

What is the capital of Arizona?

How many states border California?

What is the largest state?

Prepositions

Querying Databases

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4
AZ	Phoenix	2.7

Border

State1	State2
WA	OR
WA	ID
CA	OR
CA	NV

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

What is the capital of Arizona?

How many states border California?

What is the largest state?

Superlatives

Querying Databases

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4
AZ	Phoenix	2.7

Border

State1	State2
WA	OR
WA	ID
CA	OR
CA	NV

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

What is the capital of Arizona?

How many states border California?

What is the largest state?

Determiners

Querying Databases

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4
AZ	Phoenix	2.7

Border

State1	State2
WA	OR
WA	ID
CA	OR
CA	NV

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

What is the capital of Arizona?

How many states border California?

What is the largest state?

Questions

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

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

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

Noun phrases name specific entities

Washington

WA

Florida

The Sunshine State

FL

Noun Phrases

State

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

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

e-typed entities

Noun phrases name specific entities

Washington

WA

WA

Florida

The Sunshine State

FL

FL

Noun Phrases

State

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

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

Noun phrases name specific entities

$$\frac{\text{Washington}}{NP \atop WA}$$

$$\frac{\text{The Sunshine State}}{NP \atop FL}$$

Verb Relations

State

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

Border

State1	State2
WA	OR
WA	ID
CA	OR
CA	NV

Verbs express relations between entities

Nevada **borders** California

$border(NV, CA)$

Verb Relations

State

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

Border

State1	State2
WA	OR
WA	ID
CA	OR
CA	NV

Verbs express relations between entities

Nevada **borders** California

$border(NV, CA)$

$true$

Verb Relations

State

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

$$
\begin{array}{ccc}
\text{Nevada} & \text{borders} & \text{California} \\
\hline
NP & S\backslash NP/NP & NP \\
NV & \lambda x.\lambda y.border(y,x) & CA
\end{array}
$$

$$
\frac{S\backslash NP/NP : \lambda x.\lambda y.border(y,x) \quad NP : CA}{S\backslash NP : \lambda y.border(y,CA)} >
$$

$$
\frac{NP : NV \quad S\backslash NP : \lambda y.border(y,CA)}{S : border(NV,CA)} <
$$

Nouns

State

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

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

Nouns are functions that define entity type

state

$\lambda x.state(x)$

mountain

$\lambda x.mountain(x)$

Nouns

State

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

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

Nouns are functions that define entity type

state

$\lambda x.state(x)$

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

mountain

$\lambda x.mountain(x)$

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

$e \rightarrow t$

functions define sets

Nouns

State

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

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

Nouns are functions that define entity type

$$\frac{\textbf{state}}{N : \lambda x.state(x)}$$

$$\frac{\textbf{mountain}}{N : \lambda x.mountain(x)}$$

Prepositions

State

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

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

Prepositional phrases are conjunctive modifiers

mountain in Colorado

Prepositions

State

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

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

Prepositional phrases are conjunctive modifiers

mountain

$\lambda x.mountain(x)$

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

Prepositions

State

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

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

Prepositional phrases are conjunctive modifiers

mountain in Colorado

$$\lambda x.mountain(x) \land in(x, CO)$$

{ BIANCA , ANTERO }

Prepositions

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

mountain: N, $\lambda x.mountain(x)$

in: PP/NP, $\lambda y.\lambda x.in(x,y)$

Colorado: NP, CO

$>$: PP, $\lambda x.in(x,CO)$

$N\backslash N$, $\lambda f.\lambda x.f(x) \wedge in(x,CO)$

$<$: N, $\lambda x.mountain(x) \wedge in(x,CO)$

Function Words

State

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

Border

State1	State2
WA	OR
WA	ID
CA	OR
CA	NV

Certain words are used to modify syntactic roles

state **that** borders California

$\lambda x.state(x) \wedge border(x, CA)$

{ OR , NV , AZ }

Function Words

State

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

state	that	borders	California
N	$PP/(S\backslash NP)$	$S\backslash NP/NP$	NP
NV	$\lambda f.f$	$\lambda x.\lambda y.border(y,x)$	CA

$$\xrightarrow{>}\quad S\backslash NP \quad \lambda y.border(y, CA)$$

$$\xrightarrow{>}\quad PP \quad \lambda y.border(y, CA)$$

$$N\backslash N \quad \lambda f.\lambda y.f(y) \wedge border(y, CA)$$

$$\xrightarrow{<}\quad N \quad \lambda x.state(x) \wedge (x, CA)$$

Function Words

State

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

Border

State1	State2
WA	OR
WA	ID
CA	OR
CA	NV

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

State

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

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

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

$$\iota : (e \to t) \to e$$

the mountain in Washington

Definite Determiners

State

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

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

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

$$\iota : (e \rightarrow t) \rightarrow e$$

mountain in Washington

$$\lambda x.mountain(x) \wedge in(x, WA)$$

$$\left\{ \text{RAINIER} \right\}$$

Definite Determiners

State

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

Mountains

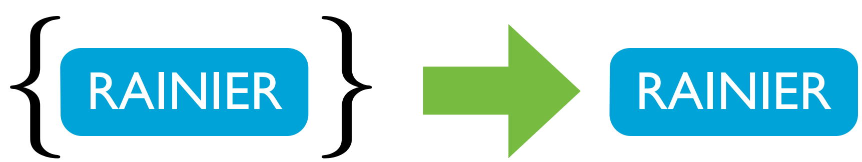
Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

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

$$\iota : (e \rightarrow t) \rightarrow e$$

the mountain in Washington

$$\iota x.mountain(x) \wedge in(x, WA)$$



Definite Determiners

State

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

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

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

$$\iota : (e \rightarrow t) \rightarrow e$$

the mountain in Colorado

$$\iota x. mountain(x) \wedge in(x, CO)$$

{ BIANCA , ANTERO } → ?

Definite Determiners

State

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

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

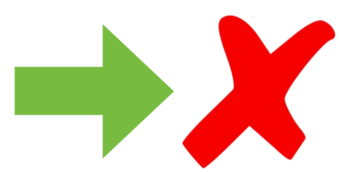
Definite determiner selects the single members of a set when such exists

$$\iota : (e \rightarrow t) \rightarrow e$$

the mountain in Colorado

$$\iota x.mountain(x) \land in(x, CO)$$

{ BIANCA , ANTERO }



No information to disambiguate

Definite Determiners

State

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

the $\qquad$ mountain in Colorado

$$\frac{\text{the}}{\begin{array}{c} NP/N \\ \lambda f.\iota x.f(x) \end{array}} \qquad \frac{\text{mountain in Colorado} \\ \vdots}{\begin{array}{c} N \\ \lambda x.mountain(x) \wedge in(x, CO) \end{array}}$$

$$\xrightarrow{\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad >}$$

$$\begin{array}{c} NP \\ \iota x.mountain(x) \wedge in(x, CO) \end{array}$$

Indefinite Determiners

State

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

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

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

$$\mathcal{A} : (e \rightarrow t) \rightarrow e$$

state with a mountain

$$\lambda x.state(x) \wedge in(\mathcal{A}y.mountain(y), x)$$

[Steedman 2011; Artzi and Zettlemoyer 2013b]

Indefinite Determiners

State

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

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA
Shasta	CA

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

$$\mathcal{A} : (e \rightarrow t) \rightarrow e$$

state with a mountain

$$\lambda x.state(x) \wedge in(\mathcal{A}y.mountain(y), x)$$

$$\Updownarrow$$

$$\lambda x.state(x) \wedge \exists y.mountain(y) \wedge in(y, x)$$

Exists

[Steedman 2011; Artzi and Zettlemoyer 2013b]

Indefinite Determiners

state	with	a	mountain
N	PP/NP	NP/N	N
$\lambda x.state(x)$	$\lambda x.\lambda y.in(x,y)$	$\lambda f.\mathcal{A}x.f(x)$	$\lambda x.mountain(x)$

$$\frac{NP/N \quad N}{NP : \mathcal{A}x.mountain(x)} >$$

$$\frac{PP/NP \quad NP}{PP : \lambda y.(\mathcal{A}x.mountain(x), y)} >$$

$$\frac{PP}{N\backslash N : \lambda f.\lambda y.f(y) \land (\mathcal{A}x.mountain(x), y)}$$

$$\frac{N \quad N\backslash N}{N : \lambda y.state(y) \land (\mathcal{A}x.mountain(x), y)} <$$

Superlatives

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4
AZ	Phoenix	2.7
WA	Olympia	4.1
NY	Albany	17.5
IL	Springfield	11.4

Superlatives select optimal entities according to a measure

the largest state

$$argmax(\lambda x.state(x), \lambda y.pop(y))$$

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

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

AL	3.9
AK	0.4
Seattle	2.7
San Francisco	4.1
NY	17.5
IL	11.4

Superlatives

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4
AZ	Phoenix	2.7
WA	Olympia	4.1
NY	Albany	17.5
IL	Springfield	11.4

Superlatives select optimal entities according to a measure

the largest state

$$argmax(\lambda x.state(x), \lambda y.pop(y))$$

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

CA

AL	3.9
AK	0.4
Seattle	2.7
San Francisco	4.1
NY	17.5
IL	11.4

Superlatives

State

Abbr.	Capi
AL	Montgo
AK	Junea
AZ	Phoe
WA	Olym
NY	Alba
IL	Springf

$$\dfrac{\text{the largest}}{\begin{array}{c} NP/N \\ \lambda f.argmax(\lambda x.f(x), \lambda y.pop(y)) \end{array}} \quad \dfrac{\text{state}}{\begin{array}{c} N \\ \lambda x.state(x) \end{array}}$$

$$\xrightarrow{\hspace{20em}} >$$

$$\begin{array}{c} NP \\ argmax(\lambda x.state(x), \lambda y.pop(y)) \end{array}$$

Superlatives

the most	populated	state
$NP/N/N$	N	N
$\lambda g.\lambda f.argmax(\lambda x.f(x), \lambda y.g(y))$	$\lambda x.pop(x)$	$\lambda x.state(x)$

$$\xrightarrow{>} \quad NP/N \quad \lambda f.argmax(\lambda x.f(x), \lambda y.pop(y))$$

$$\xrightarrow{>} \quad NP \quad argmax(\lambda x.state(x), \lambda y.pop(y))$$

Representing Questions

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4

Border

State1	State2
WA	OR
WA	ID
CA	OR

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA

Which mountains are in Arizona?

Represent questions as the queries that generate their answers

Representing Questions

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4

Border

State1	State2
WA	OR
WA	ID
CA	OR

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA

Which mountains are in Arizona?

$$\text{SELECT Name FROM Mountains WHERE State == AZ}$$

Represent questions as the queries that generate their answers

Reflects the query SQL

Representing Questions

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4

Border

State1	State2
WA	OR
WA	ID
CA	OR

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA

Which mountains are in Arizona?

$$\lambda x.mountain(x) \wedge in(x, AZ)$$

Represent questions as the queries that generate their answers

Reflects the query SQL

Representing Questions

State

Abbr.	Capital	Pop.
AL	Montgomery	3.9
AK	Juneau	0.4

Border

State1	State2
WA	OR
WA	ID
CA	OR

Mountains

Name	State
Bianca	CO
Antero	CO
Rainier	WA

How many states border California?

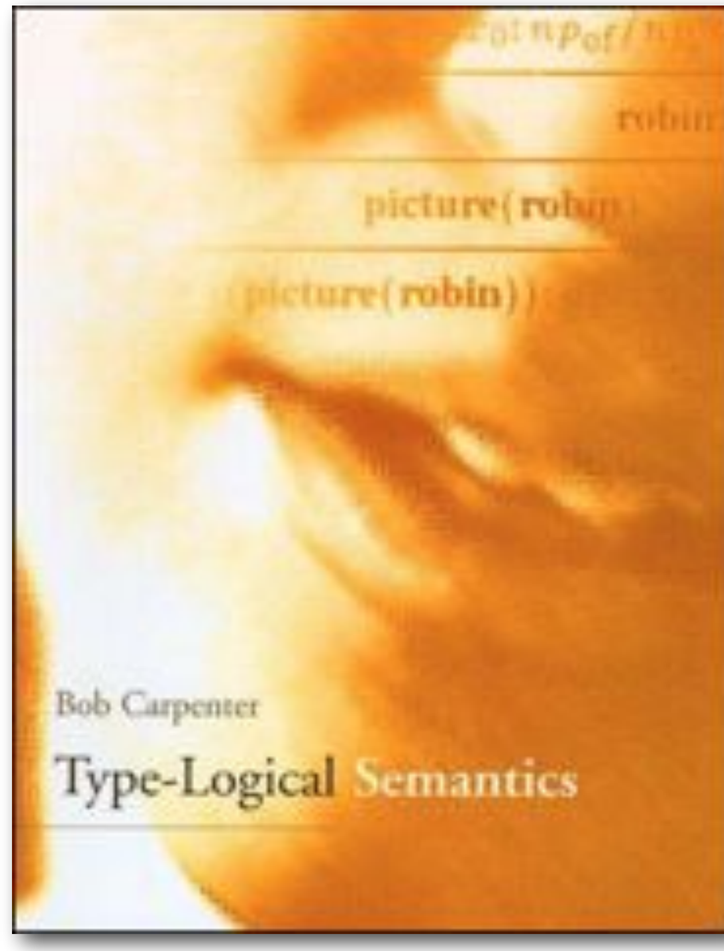
$$count(\lambda x.state(x) \wedge border(x, CA))$$

Represent questions as the queries that generate their answers

Reflects the query SQL

More Reading about Modeling

Type-Logical Semantics
by Bob Carpenter



[Carpenter 1997]

Today

Parsing	Combinatory Categorial Grammars
Learning	Unified learning algorithm
Modeling	Best practices for semantics design

[fin]