Semantic Parsing with Combinatory Categorial Grammars

Yoav Artzi, Nicholas FitzGerald and Luke Zettlemoyer
University of Washington

Website http://yoavartzi.com/tutorial

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

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

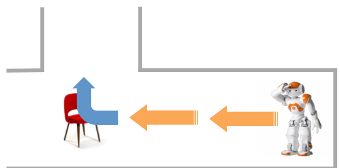
Recover **complete** meaning representation

More informative

Example Task

Instructing a Robot

at the chair, turn right



Language to Meaning

Semantic Parsing

Recover **complete** meaning representation

More informative

<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

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

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

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

Learn

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

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]

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

Parsing | Learning | Modeling

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

Parsing | Learning | Modeling

- 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]

Parsing | Learning | Modeling

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

Lambda Calculus

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

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

[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: $located\_in$ — Logical expression

Arguments: $AUSTIN, TEXAS$ — 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 operator: λ

Variable: y

Body: $located\_in(x, y)$

Lambda Calculus
Quantifiers?

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

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

Lambda Calculus
Syntactic Sugar

$$\wedge(A, \wedge(B, C)) \Leftrightarrow A \wedge B \wedge C$$
$$\vee(A, \vee(B, C)) \Leftrightarrow A \vee B \vee C$$
$$\neg(A) \Leftrightarrow \neg A$$
$$\mathcal{Q}(\lambda x.f(x)) \Leftrightarrow \mathcal{Q}x.f(x)$$
$$\text{for } \mathcal{Q} \in \{\iota, \mathcal{A}, \exists, \forall\}$$

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

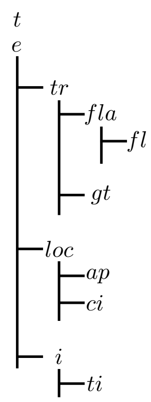
t Truth-value
e Entity

- Simple types
- Complex types

$< e, t >$ (Type constructor)

$<< e, t >, e >$ — $< e, t >$: Domain; e: Range

Lambda Calculus
Typing



- Simple types
- Complex types

$< e, t >$ (Type constructor)

$<< e, t >, e >$ — $< e, t >$: Domain; e: Range

- Hierarchical typing system

Simply Typed Lambda Calculus

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

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...	

Show me mountains in states bordering Texas

[Zettlemoyer and Collins 2005]

Capturing Meaning with Lambda Calculus

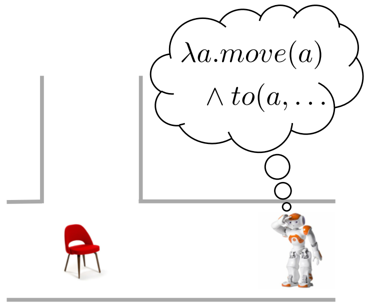
SYSTEM	how can I help you ?
USER	i ' d like to fly to new york
SYSTEM	flying to new york . leaving what city ?
USER	from boston on june seven with american airlines
SYSTEM	flying to new york . what date would you like to depart boston ?
USER	june seventh
SYSTEM	do you have a preferred airline ?
USER	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 later

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

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

$$\dfrac{S\backslash NP/ADJ \;\; ADJ}{S\backslash NP : \lambda x.fun(x)} >$$

$$\dfrac{NP \;\; S\backslash NP}{S : fun(CCG)} <$$

[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

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

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

Use lexicon to match words and phrases with their categories

CCG Operations

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

CCG Operations
Application

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

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

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

CCG Operations
Application

Argument	Function		Result	
$B : g$	$A\backslash B : f$	$\Rightarrow$	$A : f(g)$	$(<)$

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

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

Parsing with CCGs

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

Use lexicon to match words and phrases with their categories

Parsing with CCGs

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

Combine categories using operators

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

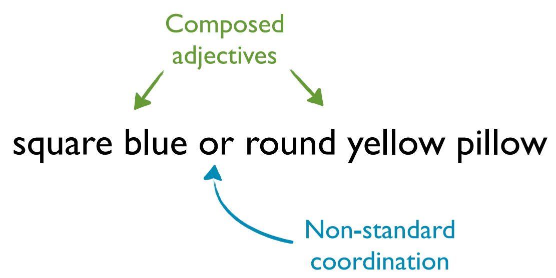
Parsing with CCGs

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

Combine categories using operators

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

Parsing with CCGs



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

$$\underbrace{A/B : f}_{f} \quad \underbrace{B/C : g}_{g} \Rightarrow \underbrace{A/C : \lambda x.f(g(x))}_{f \circ g} \quad (> B)$$

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

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

* Formal definition of logical composition in supplementary slides

CCG Operations
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

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 $\lambda x.pillow(x)$

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

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

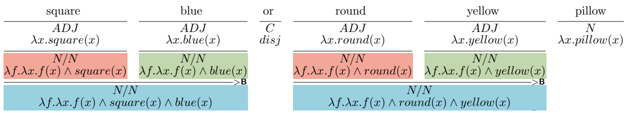
Parsing with CCGs

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

Shift adjectives to combine

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

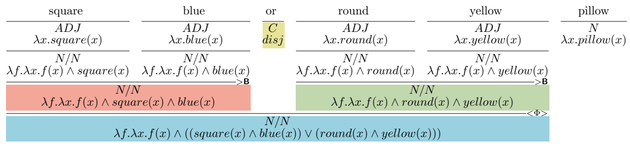
Parsing with CCGs



Compose pairs of adjectives

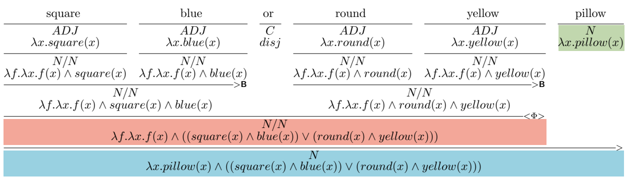
$$A/B : f \quad B/C : g \Rightarrow A/C : \lambda x.f(g(x)) \quad (> B)$$

Parsing with CCGs



Coordinate composed adjectives

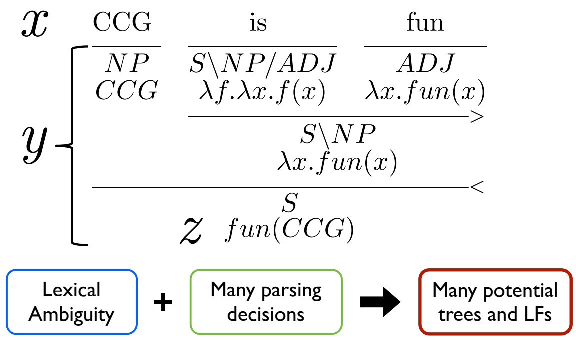
Parsing with CCGs



Apply coordinated adjectives to noun

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

Parsing with CCGs



Weighted Linear CCGs

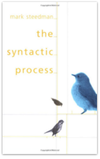
- Given a weighted linear model:
  - CCG lexicon Λ
  - Feature function $f : X \times Y \rightarrow \mathbb{R}^m$
  - Weights $w \in \mathbb{R}^m$
- The best parse is:

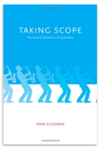
$$y^* = \arg\max_y w \cdot f(x, y)$$

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

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 — house $\vdash ADJ : \lambda x.of(x, \iota y.house(y))$

the dog of the house — house $\vdash N : \lambda x.house(x)$

$\iota x.dog(x) \wedge of(x, \iota y.house(y))$

the garden dog — garden $\vdash ADJ : \lambda x.of(x, \iota y.garden(y))$

$\iota x.dog(x) \wedge of(x, \iota y.garden(y))$

- Lexical entries share information
- Decomposition of entries can lead to more compact lexicons

Factored Lexicons

the house dog — house $\vdash ADJ : \lambda x.of(x, \iota y.house(y))$

the dog of the house — house $\vdash N : \lambda x.house(x)$

$\iota x.dog(x) \wedge of(x, \iota y.house(y))$

the garden dog — garden $\vdash ADJ : \lambda x.of(x, \iota y.garden(y))$

$\iota x.dog(x) \wedge of(x, \iota y.garden(y))$

- Lexical entries share information
- Decomposition of entries can lead to more compact lexicons

Factored Lexicons

house $\vdash ADJ : \lambda x.of(x, \iota y.house(y))$

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

garden $\vdash ADJ : \lambda x.of(x, \iota y.garden(y))$

⇒

Lexemes

(garden, $\{garden\}$)

(house, $\{house\}$)

Templates

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

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

Factored Lexicons

Templates

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

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

- Capture systematic variations in word usage
- Each variation can then be applied to compact units of lexical meaning

Lexemes

(garden, $\{garden\}$)

(house, $\{house\}$)

- Model word meaning
- Abstracts the compositional nature of the word

Factored Lexicons

$(\text{garden}, \{garden\})$

Words ↕ Constants ↗

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

⬇ $\omega \leftarrow \text{garden}$
$v_1 \leftarrow garden$

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

Factored Lexicons

Original Lexicon

$\text{flight} \vdash S|NP : \lambda x.flight(x)$
$\text{flight} \vdash S|NP/(S|NP) : \lambda f.\lambda x.flight(x) \land f(x)$
$\text{flight} \vdash S|NP\backslash(S|NP) : \lambda f.\lambda x.flight(x) \land f(x)$
$\text{ground transport} \vdash S|NP : \lambda x.trans(x)$
$\text{ground transport} \vdash S|NP/(S|NP) : \lambda f.\lambda x.trans(x) \land f(x)$
$\text{ground transport} \vdash S|NP\backslash(S|NP) : \lambda f.\lambda x.trans(x) \land f(x)$

Factored Lexicon

$(\text{flight}, \{flight\})$
$(\text{ground transport}, \{trans\})$

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

Factoring a Lexical Entry

$\text{house} \vdash ADJ : \lambda x.of(x, \iota y.house(y))$

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

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

Maximal factoring
$(\text{house}, \{of, house\})$
$\lambda(\omega, \{v_i\}_1^n).[\omega \vdash ADJ : \lambda x.v_1(x, \iota y.v_2(y))]$

Parsing | Learning | Modeling

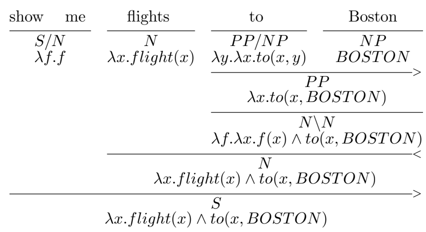
- Lambda calculus
- Parsing with Combinatory Categorial Grammars
- Linear CCGs
- Factored lexicons

Learning

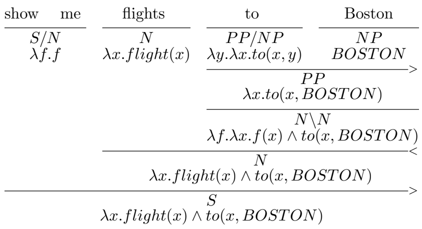
Data → Learning Algorithm → CCG

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

Parsing as Structure Prediction



Learning CCG



Lexicon | Combinators (Predefined) | w

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

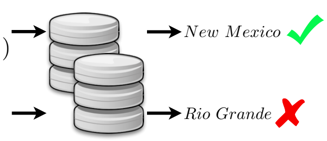
Learning from Query Answers

What is the largest state that borders Texas?

$New\ Mexico$

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

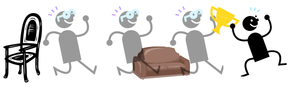
$argmax(\lambda x.river(x) \wedge in(x, TX), \lambda y.size(y))$



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

Weak Supervision
Learning from 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

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

For $t = 1 \dots T$: [iterate epochs]

For $i = 1 \dots n$: [iterate examples]

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

If $y^* \neq y_i$: [check]

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

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

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

$$update_i = f(x_i, y_i) - E_{p(y|x_i)} f(x_i, y)$$

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

$$update_i = f(x_i, y_i) - f(x_i, y^*) \text{ where } y^* = \arg\max_y w \cdot f(x_i, y)$$

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

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

Step 2: Use online, stochastic gradient updates, for example *i*:

$$update_i = E_{p(y_i,h|x_i)}[f(x_i,h,y_i)] - E_{p(y,h|x_i)}[f(x_i,h,y)]$$

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

$$update_i = f(x_i,h',y_i) - f(x_i,h^*,y^*) \text{ where}$$

$y^*, h^* = \arg\max_{y,h} w \cdot f(x_i,h,y)$ and $h' = \arg\max_h w \cdot f(x_i,h,y_i)$

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]

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

If $y^* \neq y_i$: [check]

$h' \leftarrow \arg\max_h \langle \theta, \Phi(x_i, h, y_i) \rangle$ [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

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

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

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

Output: Parameters θ and lexicon Λ

- Online
- Input:
  $\{(x_i, \mathcal{V}_i) : i = 1 \ldots n\}$
- 2 steps:
  - Lexical generation
  - Parameter update

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

Initialize parameters and lexicon

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

Step 1: (Lexical generation)

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

θ weights

Λ_0 initial lexicon

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

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

Iterate over data

Step 1: (Lexical generation)

Step 2: (Update parameters)

Output: Parameters θ and lexicon Λ

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

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

Generate a large set of potential lexical entries

θ weights

x_i sentence

$\mathcal{V}_i$ validation function

$GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$
lexical generation function

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

Generate a large set of potential lexical entries

θ weights

x_i sentence

$\mathcal{V}_i$ validation function

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

Generate a large set of potential lexical entries

θ weights

x_i sentence

$\mathcal{V}_i$ validation function

$GENLEX(x_i, \mathcal{V}_i; \Lambda, \theta)$
lexical generation function

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

$\mathcal{Y}$ all parses

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

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

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_i sentence

k beam size

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

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 lexical entries from highest scoring valid parses

θ weights

$\mathcal{V}$ validation function

$LEX(y)$ set of lexical entries

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

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

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

Update model's 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)

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

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

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

θ weights

x_i sentence

$\mathcal{V}_i$ validation function

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

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

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

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

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

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

$\mathcal{V}$ validation function

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

lexical generation function

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

Unified Learning Algorithm

Supervised

$\mathcal{V}$

Template-based $GENLEX$

Unification-based $GENLEX$

Weakly Supervised

$\mathcal{V}$

Template-based $GENLEX$

Supervised Learning

show me the afternoon flights from LA to boston

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

Parse structure is latent

Supervised Learning

Supervised

$\mathcal{V}$

Template-based $GENLEX$

Unification-based $GENLEX$

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 → x
Logical form → z
Lexicon → Λ
Weights → θ

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
...

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
...

$flight$
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
...

× $flight$, to, NYC

Create lexemes

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

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
...

× $flight$, to, NYC

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

Initialize templates

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

Fast Parsing with Pruning

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

I want a flight from New York to Boston on Delta

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

Fast Parsing with Pruning

I want a flight from New York to Boston on Delta

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

...	from	New York	to	Boston	...
	PP/NP $\lambda x.\lambda y.to(y, x)$	NP NYC	PP/NP $\lambda x.\lambda y.to(y, x)$	NP BOS	
	✗ PP $\lambda y.to(y, NYC)$ (>)		PP $\lambda y.to(y, BOS)$ (>)		

Fast Parsing with Pruning

I want a flight from New York to Boston on Delta

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

...	from	New York	to	Boston	...
	PP/NP $\lambda x.\lambda y.to(y,x)$	NP NYC	PP/NP $\lambda x.\lambda y.to(y,x)$	NP BOS	
	PP $\lambda y.to(y, NYC)$ ✗		PP $\lambda y.to(y, BOS)$		
			$N\backslash N$ $\lambda f.\lambda y.f(y) \wedge to(y, BOS)$		

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

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

Initialize the lexicon with:

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

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 CCG categories:
  1. Split logical form h to f and g s.t.

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

  2. Infer syntax from logical form type

$S : \lambda x.flight(x) \wedge to(x, BOS)$ ➡

$S/(S|NP) : \lambda f.\lambda x.flight(x) \wedge f(x)$
$S|NP : \lambda x.to(x, BOS)$

$S/NP : \lambda y.\lambda x.flight(x) \wedge f(x, y)$
$NP : BOS$

...

Unification-based GENLEX

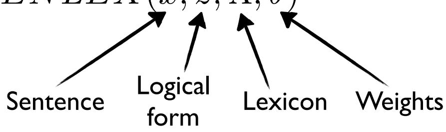
- Split text and create all pairs

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

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

Unification-based

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

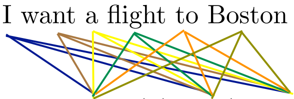


1. Find highest scoring correct parse
2. Find split that most increases score
3. Return new lexical entries

Parameter Initialization

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

I want a flight to Boston



$S : \lambda x.flight(x) \wedge to(x, BOS)$

Initial score for new lexical entries: average over pairwise weights

Unification-based

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

I want a flight to Boston
$\lambda x.flight(x) \wedge to(x, BOS)$

Unification-based

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

I want a flight to Boston
$\lambda x.flight(x) \wedge to(x, BOS)$

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

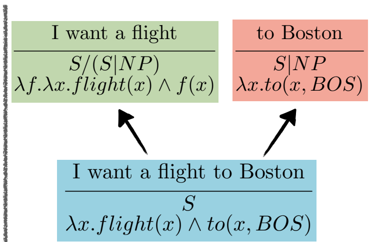
$$\frac{\text{I want a flight to Boston}}{\begin{array}{c}S\\ \lambda x.flight(x) \wedge to(x, BOS)\end{array}}$$

Unification-based

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

I want a flight to Boston
$\lambda x.flight(x) \wedge to(x, BOS)$

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



Unification-based
$GENLEX(x, z; \Lambda, \theta)$

I want a flight to Boston
$\lambda x.flight(x) \wedge to(x, BOS)$

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

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

to Boston: $S|NP$: $\lambda x.to(x, BOS)$

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

Unification-based
$GENLEX(x, z; \Lambda, \theta)$

I want a flight to Boston
$\lambda x.flight(x) \wedge to(x, BOS)$

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

Iteration 2

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

to Boston: $S|NP$: $\lambda x.to(x, BOS)$

$\rightarrow$ S : $\lambda x.flight(x) \wedge to(x, BOS)$

Unification-based
$GENLEX(x, z; \Lambda, \theta)$

I want a flight to Boston
$\lambda x.flight(x) \wedge to(x, BOS)$

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

Iteration 2

to: $(S|NP)/NP$: $\lambda y.\lambda x.to(x, y)$

Boston: NP : BOS

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

to Boston: $S|NP$: $\lambda x.to(x, BOS)$

$\rightarrow$ S : $\lambda x.flight(x) \wedge to(x, BOS)$

Unification-based
$GENLEX(x, z; \Lambda, \theta)$

I want a flight to Boston
$\lambda x.flight(x) \wedge to(x, BOS)$

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

Iteration 2

to: $(S|NP)/NP$: $\lambda y.\lambda x.to(x, y)$

Boston: NP : BOS

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

to Boston: $S|NP$: $\lambda x.to(x, BOS)$

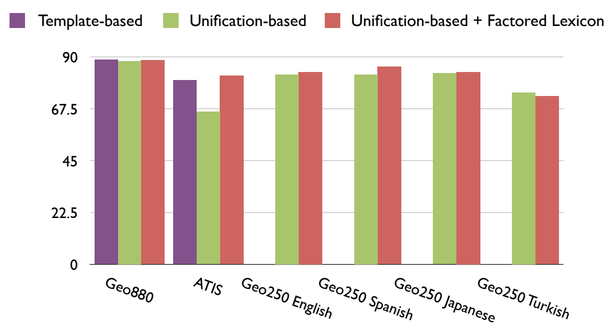
$\rightarrow$ S : $\lambda x.flight(x) \wedge to(x, BOS)$

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	✓	?

Recap

CCGs

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

$$\dfrac{\begin{matrix} NP \\ CCG \end{matrix} \quad \dfrac{\begin{matrix} S\backslash NP/ADJ \\ \lambda f.\lambda x.f(x) \end{matrix} \quad \begin{matrix} ADJ \\ \lambda x.fun(x) \end{matrix}}{\begin{matrix} S\backslash NP \\ \lambda x.fun(x) \end{matrix}} >}{\begin{matrix} S \\ fun(CCG) \end{matrix}} <$$

[Steedman 1996, 2000]

Recap

Unified Learning Algorithm

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

- 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

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

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

Unified Learning Algorithm

Supervised

$\mathcal{V}$

Template-based $GENLEX$

Unification-based $GENLEX$

Weakly Supervised

$\mathcal{V}$

Template-based $GENLEX$

Weak Supervision

What is the largest state that borders Texas?

New Mexico

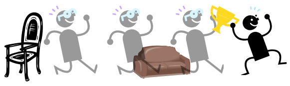
[Clarke et al. 2010; Liang et al. 2011]

Weak Supervision

What is the largest state that borders Texas?

New Mexico

at the chair, move forward three steps past the sofa



Execute the logical form and observe the result

Weakly Supervised Validation Function

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

$y \in \mathcal{Y}$ parse

$e_i \in \mathcal{E}$ available execution result

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

logical form at the root of y

[Artzi and Zettlemoyer 2013b]

Weakly Supervised Validation Function

Depends on supervision

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

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

$y \in \mathcal{Y}$ parse

$e_i \in \mathcal{E}$ available execution result

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

logical form at the root of y

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

Weakly Supervised Validation Function

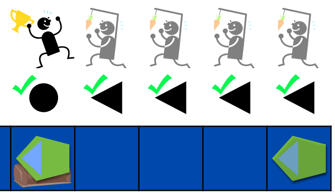
Example $EXEC(y)$:

Robot moving in an environment

Example supervision:

Complete Demonstration

Validate all steps



Weakly Supervised Validation Function

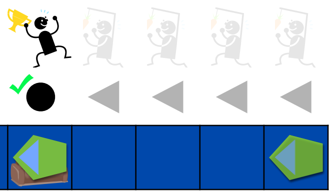
Example $EXEC(y)$:

Robot moving in an environment

Example supervision:

Final State

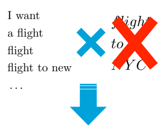
Validate only last position



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

I want a flight to new york

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



No access to labeled logical form

(flight, $\{flight\}$)

(I want, $\{\}$)

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

...

Initialize templates

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

I want $\vdash S/NP : \lambda x.x$

flight to new : $S\backslash NP/NP : \lambda x.\lambda y.to(x, y)$

...

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

I want a flight to new york

I want / a flight / flight / flight to new / … ✕ $flight, from, to, ground\_transport, dtime, atime, NYC, BOS, LA, SEA, \ldots$

Use all logical constants in the system instead

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

Many more lexemes

Initialize templates

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

Huge number of lexical entries

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

I want a flight to new york

I want / a flight / flight / flight to new / … ✕ $flight, from, to, ground\_transport, dtime, atime, NYC, BOS, LA, SEA, \ldots$

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

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

Huge number of lexical entries

Parse to prune generated lexicon

Intractable

Model

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, $\{to, NYC\}$)
…

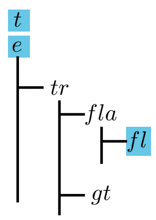
Initialize templates

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

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

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

Coarse Ontology



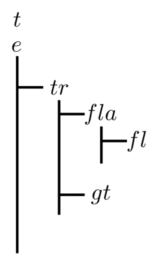
$flight_{<fl,t>}, from_{<fl,<loc,t>>}, to_{<fl,<loc,t>>},$
$ground\_transport_{<gt,t>}, dtime_{<tr,<ti,t>>}, atime_{<tr,<ti,t>>},$
$NYC_{ci}, BOS_{ci}, JFK_{ap}, LAS_{ap}, \ldots$

Generalize types

$flight_{<e,t>}, from_{<e,<e,t>>}, to_{<e,<e,t>>},$
$ground\_transport_{<e,t>}, dtime_{<e,<e,t>>}, atime_{<e,<e,t>>},$
$NYC_{e}, BOS_{e}, LA_{e}, SEA_{e}, \ldots$

$flight_{<fl,t>}$
$fl \rightarrow e$
$t \rightarrow t$
$flight_{<e,t>}$

Coarse Ontology



$flight_{<fl,t>}, from_{<fl,<loc,t>>}, to_{<fl,<loc,t>>},$
$ground\_transport_{<gt,t>}, dtime_{<tr,<ti,t>>}, atime_{<tr,<ti,t>>},$
$NYC_{ci}, BOS_{ci}, JFK_{ap}, LAS_{ap}, \ldots$

Generalize types

$flight_{<e,t>}, from_{<e,<e,t>>}, to_{<e,<e,t>>},$
$ground\_transport_{<e,t>}, dtime_{<e,<e,t>>}, atime_{<e,<e,t>>},$
$NYC_{e}, BOS_{e}, LA_{e}, SEA_{e}, \ldots$

Merge identically typed constants

$c1_{<e,t>}, c2_{<e,<e,t>>}, c3_{e}, \ldots$

Weakly Supervised

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

I want a flight to new york

All possible sub-strings

I want	$c1_{<e,t>}$
a flight	$c2_{<e,<e,t>>}$
flight	$c3_e$
flight to new	$\ldots$
$\ldots$	

Weakly Supervised

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

I want a flight to new york

All possible sub-strings

I want	$c1_{<e,t>}$	(flight, $\{c1\}$)
a flight	$c2_{<e,<e,t>>}$	(I want, $\{\}$)
flight	$c3_e$	(flight to new, $\{c2\}$)
flight to new	$\ldots$	$\ldots$
$\ldots$		

Create lexemes

Weakly Supervised

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

I want a flight to new york

I want	$c1_{<e,t>}$
a flight	$c2_{<e,<e,t>>}$
flight	$c3_e$
flight to new	$\ldots$
$\ldots$	

(flight, $\{c1\}$)
(I want, $\{\}$)
(flight to new, $\{c2\}$)
$\ldots$

flight $\vdash N : \lambda x.c1(x)$
I want $\vdash S/NP : \lambda x.x$
flight to new $\vdash S\backslash NP/NP : \lambda x.\lambda y.c2(x, y)$
$\ldots$

Initialize templates

Weakly Supervised

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

I want a flight to new york

I want	$c1_{<e,t>}$
a flight	$c2_{<e,<e,t>>}$
flight	$c3_e$
flight to new	$\ldots$
$\ldots$	

(flight, $\{c1\}$)
(I want, $\{\}$)
(flight to new, $\{c2\}$)
$\ldots$

Coarse constants

flight $\vdash N : \lambda x.c1(x)$
I want $\vdash S/NP : \lambda x.x$
flight to new $\vdash S\backslash NP/NP : \lambda x.\lambda y.c2(x, y)$
$\ldots$

Initialize templates

Weakly Supervised

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

I want a flight to new york

flight $\vdash N : \lambda x.c1(x)$
~~I want $\vdash S/NP : \lambda x.x$~~
~~flight to new $\vdash S\backslash NP/NP : \lambda x.\lambda y.c2(x, y)$~~
$\ldots$

Prune by parsing

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 $\vdash N : \lambda x.c1(x)$
$\ldots$

Replace all coarse constants with all similarly typed constants

flight $\vdash N : \lambda x.flight(x)$
flight $\vdash N : \lambda x.ground\_transport(x)$
flight $\vdash N : \lambda x.nonstop(x)$
flight $\vdash N : \lambda x.connecting(x)$
$\ldots$

Weak Supervision Requirements

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

Experiments

Instruction:

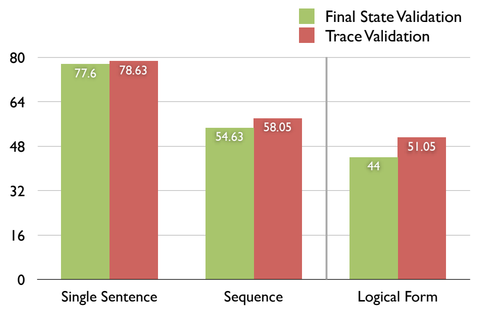
at the chair, move forward three steps past the sofa

Demonstration:

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

[Artzi and Zettlemoyer 2013b]

Results



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]

Parsing | Learning | Modeling

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

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

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

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

Washington
NP
WA

The Sunshine State
NP
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

$border(NV, CA)$

$true$

Verb Relations

State

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

Nevada — NP, NV

borders — $S\backslash NP/NP$, $\lambda x.\lambda y.border(y, x)$

California — NP, CA

$>$ $S\backslash NP$, $\lambda y.border(y, CA)$

$<$ 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
$\lambda x.state(x)$
{WA, AL, AK, ...}

$e \rightarrow t$ functions define sets

mountain
$\lambda x.mountain(x)$
{BIANCA, ANTERO, ...}

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
N
$\lambda x.state(x)$

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

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

{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: N, NV

that: $PP/(S\backslash NP)$, $\lambda f.f$

borders: $S\backslash NP/NP$, $\lambda x.\lambda y.border(y,x)$

California: NP, CA

$>$ $S\backslash NP$: $\lambda y.border(y,CA)$

$>$ PP: $\lambda y.border(y,CA)$

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

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

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

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

{ RAINIER } → RAINIER

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 Colorado

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

{ BIANCA , ANTERO } → ✗

No information to disambiguate

Definite Determiners

State

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

the: NP/N, $\lambda f.\iota x.f(x)$

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

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

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

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

a: NP/N, $\lambda f.\mathcal{A}x.f(x)$

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

$>$ NP: $\mathcal{A}x.mountain(x)$

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

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

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

Indefinite Determiners

a: $PP\backslash(PP/NP)/N$, $\lambda f.\lambda g.\lambda y.\exists x.g(x, y) \wedge f(x)$

a: $S\backslash NP\backslash(S\backslash NP/NP)/N$, $\lambda f.\lambda g.\lambda y.\exists x.g(x, y) \wedge f(x)$

a: $S\backslash(S\backslash NP)/N$, $\lambda f.\lambda g.\lambda y.\exists x.g(x, y) \wedge f(x)$

➡

a: NP/N, $\lambda f.\mathcal{A}x.f(x)$

Using the indefinite quantifier simplifies CCG handling of the indefinite determiner

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.	Cap
AL	Montgo
AK	June
AZ	Phoe
WA	Olym
NY	Alba
IL	Spring

the largest: NP/N : $\lambda f.argmax(\lambda x.f(x), \lambda y.pop(y))$

state: N : $\lambda x.state(x)$

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

Superlatives

State

Abbr.	Cap
AL	Montgo
AK	June
AZ	Phoe
WA	Olym
NY	Alba
IL	Spring

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

populated: N : $\lambda x.pop(x)$

state: N : $\lambda x.state(x)$

$>$ NP/N : $\lambda f.argmax(\lambda x.f(x), \lambda y.pop(y))$

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

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?

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

DB Queries

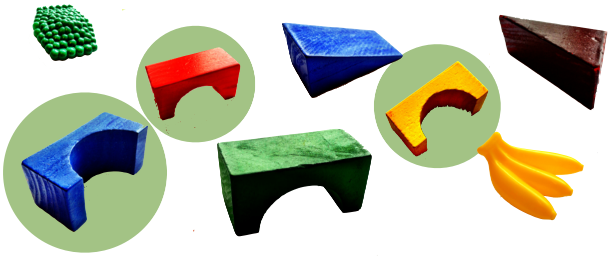
So Far

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

Next

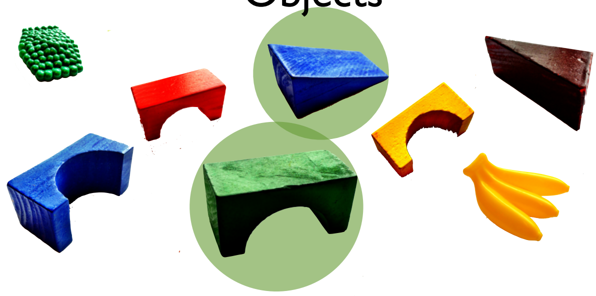
- How does this approach hold for physical objects?
- What do we need to change? Add?

Referring to Real World Objects



all the arches except the green arch

Referring to Real World Objects



the blue triangle and the green arch

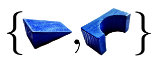
Plurality

arches
$\lambda x.arch(x)$

the arches
$\iota x.arch(x)$

Plurality

blue blocks
$\lambda x.blue(x) \wedge block(x)$



brown block
$\lambda x.brown(x) \wedge block(x)$

Plurality

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

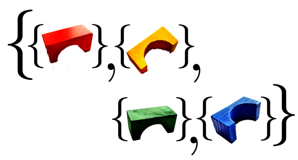
Cognitive evidence for sets being a primitive type

[Scontras et al. 2012]

Plurality

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

arch
$\lambda x.arch(x) \wedge sg(x)$

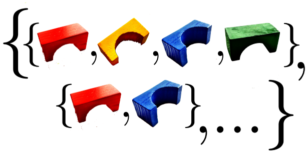


Plurality

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

arches

$\lambda x.arch(x) \wedge plu(x)$

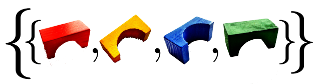


Plurality and Determiners

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

the arches

$\iota x.arch(x) \wedge plu(x)$

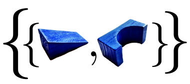


Adjectives

Adjectives are conjunctive modifiers

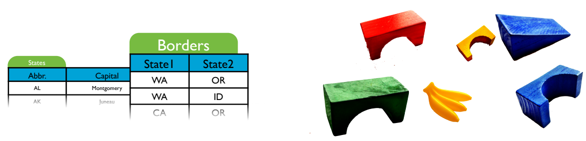
blue objects

$\lambda x.blue(x) \wedge obj(x) \wedge plu(x)$



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 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

$$do\_seq(verify(room(current\_loc)),\; move\_to(unique\_thing(\lambda x.equals(distance(x),1))),\; move\_to(right\_loc))$$

[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")

[Branavan et al. 2009, Branavan et al. 2010]

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

Noun phrases	Specific entities
Nouns	Sets of entities
Prepositional phrases Adjectives	Constrain sets

Instructions to execute

Verbs	Davidsonian events
Imperatives	Sets of events

Modeling Instructions

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

Modeling Instructions

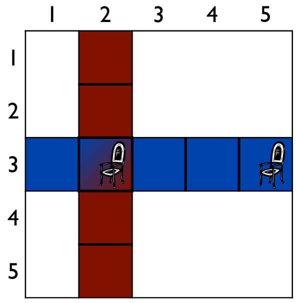
place your back against the wall of the t intersection

turn left

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

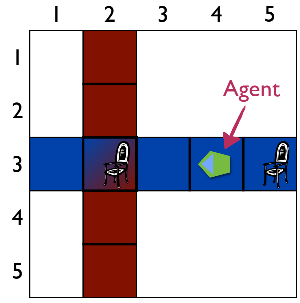
⋮

Instructional Environment



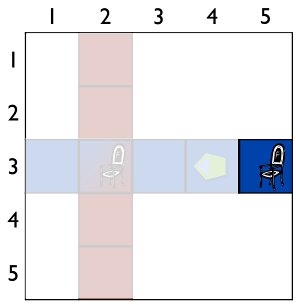
- Maps are graphs of connected positions
- Positions have properties and contain objects

Instructional Environment



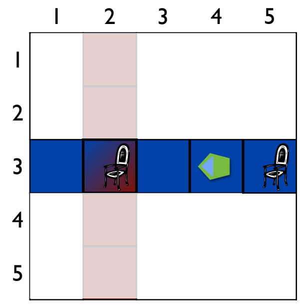
- Agent can move forward, turn right and turn left
- Agent perceives clusters of positions
- Clusters capture objects

Instructional Environment



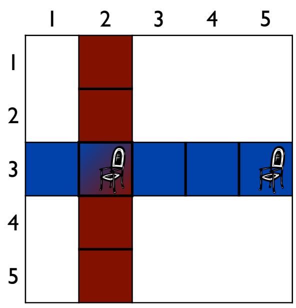
- Agent can move forward, turn right and turn left
- Agent perceives clusters of positions
- Clusters capture objects

Instructional Environment



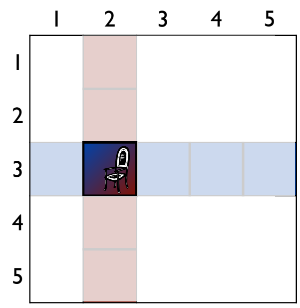
- Agent can move forward, turn right and turn left
- Agent perceives clusters of positions
- Clusters capture objects

Instructional Environment



- Refer to objects similarly to our previous domains
- "Query" the world

Grounded Resolution of Determiners

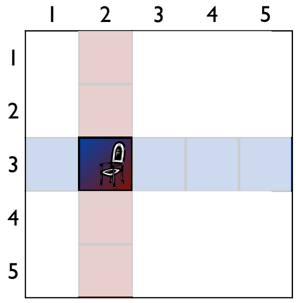


Nouns denote sets of objects

chair

$\lambda x.chair(x)$

Grounded Resolution of Determiners

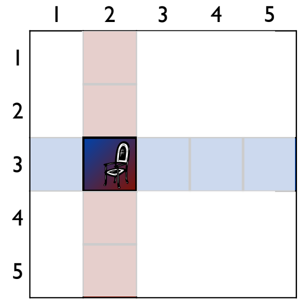


Definite determiner selects a single entity

the chair

$\iota x.chair(x)$

Grounded Resolution of Determiners

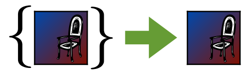


Definite determiner selects a single entity

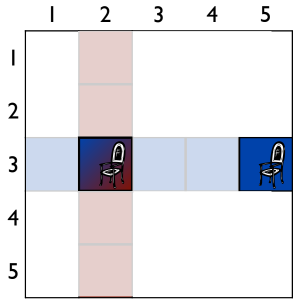
the chair

$\iota x.chair(x)$

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



Grounded Resolution of Determiners



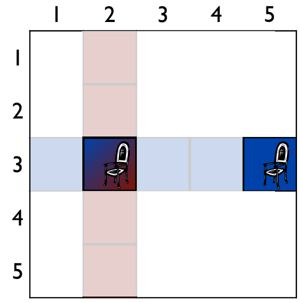
Definite determiner selects a single entity

the chair

$\iota x.chair(x)$

Fail?

Grounded Resolution of Determiners



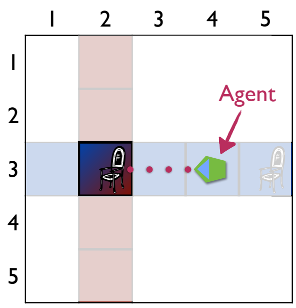
Definite determiner selects a single entity

the chair

$\iota x.chair(x)$

Must disambiguate to select a single entity

Grounded Resolution of Determiners



Definite determiner selects a single entity

the chair

$\iota x.chair(x)$

Definite determiner depends on agent state

Modeling Instructions

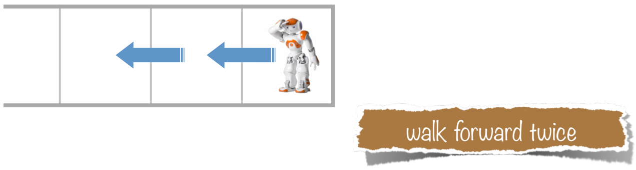
Events taking place in the world

Events refer to environment

Implicit requests

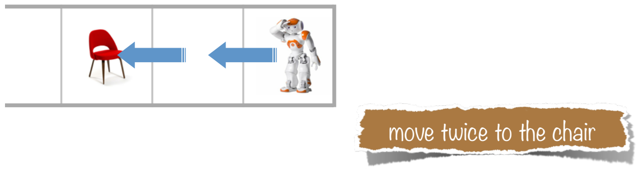
Modeling Instructions

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



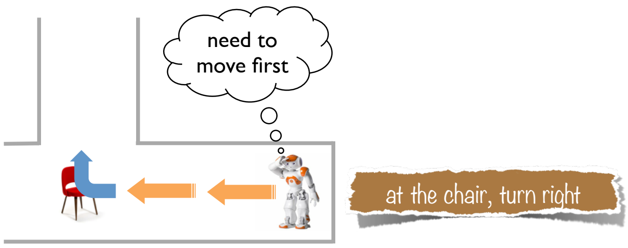
Modeling Instructions

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



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)

[Davidson 1967]

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

$$\exists a.shot(a, VINCENT, MARVIN) \wedge$$
$$in(a, \iota x.car(x)) \wedge \neg intentional(a)$$

[Davidson 1967]

Neo-Davidsonian Event Semantics

Active — Vincent shot Marvin

$$\exists a.shot(a, VINCENT, MARVIN)$$

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

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

$\exists a.shot(a, VINCENT, MARVIN)$

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

[Parsons 1990]

Neo-Davidsonian Event Semantics

Vincent shot Marvin in the car accidentally

$\exists a.shot(a) \land agent(a, VINCENT) \land$
$patient(a, MARVIN) \land in(a, \iota x.car(x)) \land \neg intentional(a)$

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

[Parsons 1990]

Neo-Davidsonian Event Semantics

$\exists a.shot(a) \land agent(a, VINCENT) \land$
$patient(a, MARVIN) \land in(a, \iota x.car(x)) \land \neg intentional(a)$

Without events:

$shot(VINCENT, MARVIN, \iota x.car(x), INTENTIONAL)$

- 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
Type	Direction	Intermediate position	Final position

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

Representing Imperatives

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

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

$$f : ev \rightarrow t$$

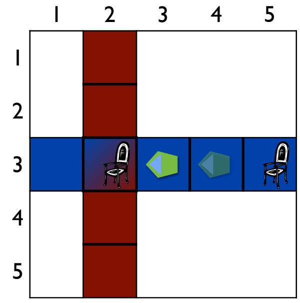
Representing Imperatives

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

Given a set, what do we **actually** execute?

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

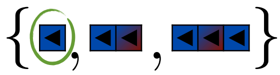
Modeling Instructions



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

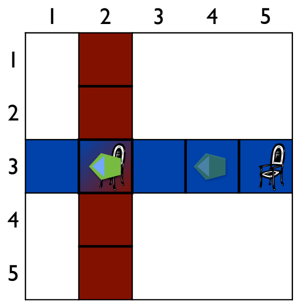
move

$\lambda a.move(a)$



Disambiguate by preferring shorter sequences

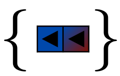
Modeling Instructions



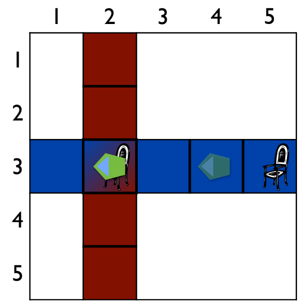
Events can be modified by adverbials

move twice

$\lambda a.move(a) \wedge len(a, 2)$



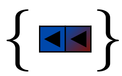
Modeling Instructions



Events can be modified by adverbials

go to the chair

$\lambda a.move(a) \wedge$
$to(a, \iota x.chair(x))$



Modeling Instructions

go	to	the	chair
S	AP/NP	NP/N	N
$\lambda a.move(a)$	$\lambda x.\lambda a.to(a,x)$	$\lambda f.\iota x.f(x)$	$\lambda x.chair(x)$

$$\frac{NP/N \quad N}{NP : \iota x.chair(x)} >$$

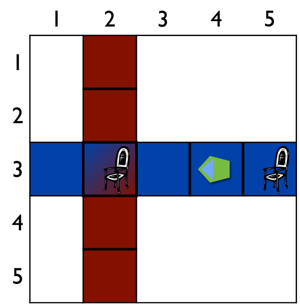
$$\frac{AP/NP \quad NP}{AP : \lambda a.to(a, \iota x.chair(x))} <$$

$$\frac{AP}{S\backslash S : \lambda f.\lambda a.f(a) \wedge to(a, \iota x.chair(x))}$$

$$\frac{S \quad S\backslash S}{S : \lambda a.move(a) \wedge to(a, \iota x.chair(x))} <$$

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

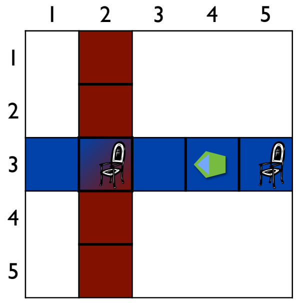
Modeling Instructions



Dynamic Models

Implicit Actions

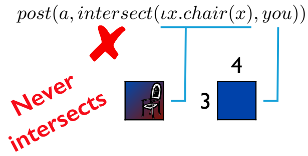
Dynamic Models



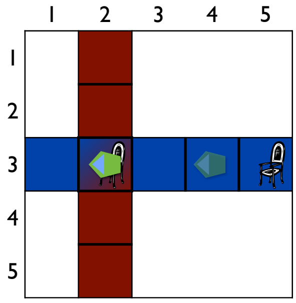
World model changes during execution

move until you reach the chair

$\lambda a.move(a) \wedge$
$post(a, intersect(\iota x.chair(x), you))$



Dynamic Models

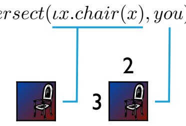


World model changes during execution

move until you reach the chair

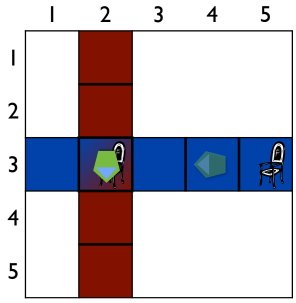
$\lambda a.move(a) \wedge$
$post(a, intersect(\iota x.chair(x), you))$

Update



Update model to reflect state change

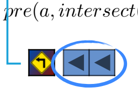
Implicit Actions



Consider action assignments with prefixed implicit actions

at the chair, turn left

$\lambda a.turn(a) \wedge dir(a, left) \wedge$
$pre(a, intersect(\iota x.chair(x), you))$



Implicit actions

Experiments

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

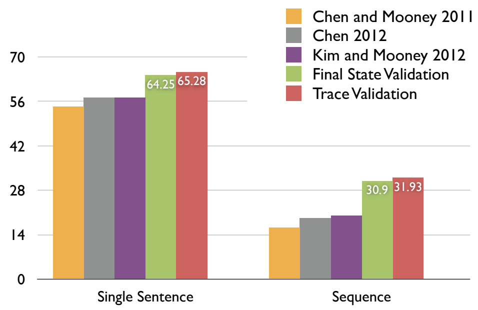
Demonstration:

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

[Artzi and Zettlemoyer 2013b]

Results

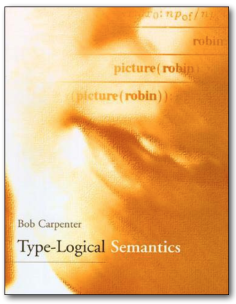
SAIL Corpus - Cross Validation



[Artzi and Zettlemoyer 2013b]

More Reading about Modeling

Type-Logical Semantics by Bob Carpenter



[Carpenter 1997]

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?
$\lambda x \,.\, \texttt{neighborhoods}(\texttt{new\_york}, x)$

How many countries use the rupee?
$\texttt{count}(x) \,.\, \texttt{countries\_used}(\texttt{rupee}, x)$

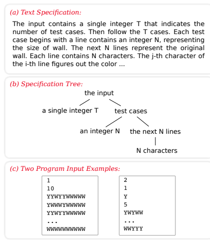
How many Peabody Award winners are there?
$\texttt{count}(x) \,.\, \exists y \,.\, \texttt{award\_honor}(y) \land \texttt{award\_winner}(y, x) \land \texttt{award}(y, \texttt{peabody\_award})$

Looking Forward: Code

Goal Program using natural language

Challenges
- Data
- Complex intent
- Complex output

See Kushman and Barzilay 2013; Lei et al. 2013



Text Description	Regular Expression
three letter word starting with 'X'	`\bX[A-Za-z]{2}\b`

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
$\lambda x.flight(x) \land from(x, bos) \land to(x, phi)$
(b) show me the ones that leave in the morning
$\lambda x.flight(x) \land from(x, bos) \land to(x, phi) \land during(x, morning)$
(c) what kind of plane is used on these flights
$\lambda y.\exists x.flight(x) \land from(x, bos) \land to(x, phi) \land during(x, morning) \land aircraft(x) = y$

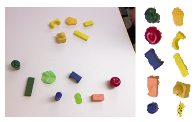
Example #2:
(a) show me flights from milwaukee to orlando
$\lambda x.flight(x) \land from(x, mil) \land to(x, orl)$
(b) cheapest
$argmin(\lambda x.flight(x) \land from(x, mil) \land to(x, orl), \lambda y.fare(y))$
(c) departing wednesday after 5 o'clock
$argmin(\lambda x.flight(x) \land from(x, mil) \land to(x, orl) \land day(x, wed) \land depart(x) > 1700\ , \lambda y.fare(y))$

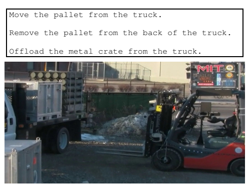
Looking Forward: Sensors

Goal Integrate semantic parsing with rich sensing on real robots

Challenges
- Data
- Managing uncertainty
- Interactive learning

See Matuszek et al. 2012; Tellex et al. 2013; Krishnamurthy and Kollar 2013





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

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