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

Language to Meaning

Information Extraction
Recover information about pre-specified relations and entities

Relation Extraction
Example Task
Relation Extraction

More informative

is$_{\text{pres}}$(OBAMA, PRESIDENT)

Language to Meaning

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

Summarization
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

Database Query
Example Task
Database Query

What states border Texas?
Oklahoma
New Mexico
Arkansas
Louisiana

Language to Meaning

Semantic Parsing
Recover complete meaning representation

Instructing a Robot
Example Task
Instructing a Robot

at the chair, turn right
Complete meaning is sufficient to complete the task

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

at the chair, move forward three steps past the sofa

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

- Grammar formalism
- Inference procedure

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

Learning

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

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

Semantic Modeling

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

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

Today

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

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

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

Parsing

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

Learning

• Supervised learning
• Weak supervision

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

Lambda Calculus

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

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

Lambda Calculus

Base Cases

• Logical constant
• Variable
• Literal
• Lambda term

[Arzi and Zettlemoyer 2013a]

[Church 1932]
Lambda Calculus
Logical Constants

• Represent objects in the world

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

Lambda Calculus
Variables

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

x, y, z, . . .

Lambda Calculus
Literals

• Represent function application

\text{city}(AUSTIN)
located_in(AUSTIN, TEXAS)

Lambda Calculus
Quantifiers?

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

∀(λx.big(x) ∧ apple(x))
¬(∃(λx.lovely(x))
∀(λx.beautiful(x) ∧ grammar(x)))
Lambda Calculus
Syntactic Sugar

\(\land (A, \land (B, C)) \Leftrightarrow A \land B \land C\)
\(\lor (A, \lor (B, C)) \Leftrightarrow A \lor B \lor C\)
\(\neg (A) \Leftrightarrow \neg A\)
\(Q(\lambda x.f(x)) \Leftrightarrow Qx.f(x)\)
for \(Q \in \{\emptyset, A, \exists, \forall\}\)

Simply Typed Lambda Calculus

- Like lambda calculus
- But, typed

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

Lambda Calculus
Typing

- Simple types
- Complex types

\(<e, t>\)
\(<<e, t>, e>\)

Simply Typed Lambda Calculus

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

Type information usually omitted

[Church 1940]

Capturing Meaning with Lambda Calculus

Mountains

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

Shasta, CA
Elbert, CO
Mountains in states bordering Texas

[Zettlemoyer and Cuffaro 2005]
Capturing Meaning with Lambda Calculus

- Flexible representation
- Can capture full complexity of natural language

More on modeling meaning later

Combinatory Categorial Grammars

<table>
<thead>
<tr>
<th>CCG</th>
<th>is</th>
<th>fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>S (\rightarrow) NP/ADJ</td>
<td>(\rightarrow) ADJ</td>
</tr>
<tr>
<td></td>
<td>(\lambda f. \lambda x. f(x))</td>
<td>(\lambda x. \text{fun}(x))</td>
</tr>
<tr>
<td></td>
<td>(\rightarrow) S (\rightarrow) NP</td>
<td>(\rightarrow) S</td>
</tr>
<tr>
<td></td>
<td>(\lambda x. \text{fun}(x))</td>
<td>(\text{fun}(\text{CCG}))</td>
</tr>
</tbody>
</table>

Combinatory Categorial Grammars

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

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

- Basic building block
- Capture syntactic and semantic information jointly

---

**CCG Categories**

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

- Basic building block
- Capture syntactic and semantic information jointly

**Semantics**

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

---

**CCG Categories**

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

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

**Semantics**

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

---

**CCG Lexical Entries**

**fun \( \mapsto ADJ : \lambda x.\text{fun}(x) \)**

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

---

**CCG Lexical Entries**

**fun \( \mapsto ADJ : \lambda x.\text{fun}(x) \)**

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

fun ⊨ ADJ : λx.fun(x)

is ⊨ (S\NP)\ADJ : λf.λx.f(x)

CCG ⊨ NP : CCG

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

Between CCGs and CFGs

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

Parsing with CCGs

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

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

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

Use lexicon to match words and phrases with their categories

Combine categories using operators

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


CCG Operations

Composition

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

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

* Formal definition of logical composition in supplementary slides
Topicalization

**CCG Operations**

- **Type Shifting**
  
  \[
  \text{Input:} \quad ADJ : \lambda x.g(x) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)
  \]

  \[
  \text{Output:} \quad PP : \lambda x.g(x) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)
  \]

  \[
  \text{AP : } \lambda e.g(e) \Rightarrow S/S : \lambda f.\lambda e.f(e) \wedge g(e)
  \]

- **Topicalization**

  \[
  \text{ADJ : } \lambda x.\text{square} \Rightarrow \text{square ADJ ADJ C ADJ ADJ N}
  \]

  \[
  \text{N/N N/N N/N N/N}
  \]

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

**Parsing with CCGs**

<table>
<thead>
<tr>
<th>square</th>
<th>blue</th>
<th>or</th>
<th>round</th>
<th>yellow</th>
<th>pillow</th>
</tr>
</thead>
<tbody>
<tr>
<td>square</td>
<td>blue</td>
<td>or</td>
<td>round</td>
<td>yellow</td>
<td>pillow</td>
</tr>
</tbody>
</table>

Use lexicon to match words and phrases with their categories

**Shift adjectives to combine**

\[
ADJ : \lambda x.g(x) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)
\]
Ambiguity

Many parsing decisions $\rightarrow$ Many potential trees and LFs

Lexical

Apply coordinated adjectives to noun

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

Coordinate composed adjectives

\[ \lambda x.f(g(x)) \]

Lexical

Coordinate composed adjectives

\[ \lambda x.f(g(x)) \]

Ambiguity

Many parsing decisions $\rightarrow$ Many potential trees and LFs

Lexical

Apply coordinated adjectives to noun

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

Lexical

Coordinate composed adjectives

\[ \lambda x.f(g(x)) \]

Lexical

Apply coordinated adjectives to noun

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

Lexical

Coordinate composed adjectives

\[ \lambda x.f(g(x)) \]

Lexical

Apply coordinated adjectives to noun

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

Lexical

Coordinate composed adjectives

\[ \lambda x.f(g(x)) \]

Lexical

Apply coordinated adjectives to noun

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

Lexical

Coordinate composed adjectives

\[ \lambda x.f(g(x)) \]
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

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

- The house dog
  house $\vdash ADJ : \lambda x. of(x, y. house(y))$
- the garden dog
  garden $\vdash ADJ : \lambda x. of(x, y. garden(y))$

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

Factored Lexicons

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

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

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

\[ \lambda(v_1,v_2)' \rightarrow \text{house} \]
\[ \lambda(x,y,z) \rightarrow \text{ADJ} : \lambda x. \text{of}(x, y, \text{house}(z)) \]

- Maximal
  - \[ [\omega \mapsto N : \lambda x. v_1(x)] \]
- Partial
  - house \[ \rightarrow \text{ADJ} : \lambda x. \text{of}(x, y, \text{house}(z)) \]
  - \[ [\omega \mapsto N : \lambda x. v_1(x)] \]

Learning

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

<table>
<thead>
<tr>
<th>show me flights to Boston</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/P/NP</td>
</tr>
<tr>
<td>λx.tol(x, BOSTON)</td>
</tr>
<tr>
<td>N</td>
</tr>
<tr>
<td>λx.flight(x)</td>
</tr>
<tr>
<td>λy.tol(x, y)</td>
</tr>
<tr>
<td>to</td>
</tr>
<tr>
<td>λx.flight(x)</td>
</tr>
<tr>
<td>λy.tol(x, y)</td>
</tr>
<tr>
<td>Boston</td>
</tr>
</tbody>
</table>

Supervised Data

Supervised learning is done from pairs of sentences and logical forms

Show me flights to Boston
λx.flight(x) ∧ tol(x, BOSTON)

I need a flight from Baltimore to Seattle
λx.flight(x) ∧ from(x, BALTIMORE) ∧ to(x, SEATTLE)

what ground transportation is available in San Francisco
λx.ground_transport(x) ∧ city(x, SF)

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

What is the largest state that borders Texas?

New Mexico

argmax(λx.state(x) ∧ border(x, TX), λy.size(y)) → New Mexico

argmax(λx.river(x) ∧ in(x, TX), λy.size(y)) → Rio Grande
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]

[Matuszek et al. 2012; Artzi and Zettlemoyer 2013b]

Weak Supervision
Learning from Conversation Logs

SYSTEM how can I help you? (OPEN_TASK)
USER i’d like to fly to new york
SYSTEM flying to new york. (CONFIRM: from(FL, ATL)) leaving what city?
(ASK: λx.λy.from(FL, ATX) ∧ y)
USER from boston on june seventh with american airlines
SYSTEM flying to new york. (CONFIRM: to(FL, NYC)) what date would you
like to depart boston? (ASK: λx.date(FL, x) ∧ to(FL, BOS))
USER june seventh

[Artzi and Zettlemoyer 2011]

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

<table>
<thead>
<tr>
<th>Data: {(x_i, y_i) : i = 1 \ldots n}</th>
</tr>
</thead>
<tbody>
<tr>
<td>For t = 1 \ldots T: [iterate epochs]</td>
</tr>
<tr>
<td>For i = 1 \ldots n: [iterate examples]</td>
</tr>
<tr>
<td>y_i^* ← \arg \max_y (\theta, \Phi(x_i, y)) [predict]</td>
</tr>
<tr>
<td>If y_i^* ≠ y_i:</td>
</tr>
<tr>
<td>θ ← θ + Φ(x_i, y_i) - Φ(x_i, y_i^*) [update]</td>
</tr>
</tbody>
</table>

[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
\[
update_i = \sum f(x_i, y_i) - E_{p(y|x)} f(x_i, y_i)
\]

Step 2: Use online, stochastic gradient updates, for example \( i \):
\[
update_i = f(x_i, y_i) - E_{p(y|x)} f(x_i, y_i)
\]

Step 3: Replace expectations with maxes (Viterbi approx.)
\[
update_i = f(x_i, y_i) - f(x_i, y^*) \text{ where } y^* = \arg \max_y w \cdot f(x_i, y)
\]

[Collins 2002]
The Perceptron with Hidden Variables

Log-linear model:

\[ p(y|x) = \sum_h p(y, h|x) \quad p(y, h|x) = e^{w \cdot f(x, h, y)} / \sum_{y', h'} e^{w \cdot f(x, h', y')} \]

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

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

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

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

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

update_i = \[ f(x_i, h^*, y^*) - f(x_i, h^*, y^*) \]

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]

\( y^*, h^* \leftarrow \arg \max_y h \{ \theta, \Phi(x_i, h, y) \} \) [predict]

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

\( h' \leftarrow \arg \max_h \{ \theta, \Phi(x_i, h, y_i) \} \) [predict hidden]

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

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

Hidden Variable Perceptron

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

Unified Learning Algorithm

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

Learning Choices

**Validation Function**

\( V : Y \rightarrow \{t, f\} \)

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

**Lexical Generation Procedure**

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

- Given:
  - sentence \( x \)
  - validation function \( V \)
  - lexicon \( \Lambda \)
  - parameters \( \theta \)
- Produce a overly general set of lexical entries

Unified Learning Algorithm

**Online**

**Input:**

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

**2 steps:**

- Lexical generation
- Parameter update
Initialize $\theta$ using $\lambda_0$, $\lambda \leftarrow \lambda_0$

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

Step 1: (Lexical generation)

a. Set $\lambda_0 \leftarrow GENLEX(x, y; \lambda, \theta)$

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

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

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

Step 2: (Update parameters)

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

---

Initialize $\theta$ using $\lambda_0$, $\lambda \leftarrow \lambda_0$

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

Step 1: (Lexical generation)

a. Set $\lambda_0 \leftarrow GENLEX(x, y; \lambda, \theta)$

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

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

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

Step 2: (Update parameters)

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

---

Initialize $\theta$ using $\lambda_0$, $\lambda \leftarrow \lambda_0$

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

Step 1: (Lexical generation)

a. Set $\lambda_0 \leftarrow GENLEX(x, y; \lambda, \theta)$

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

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

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

Step 2: (Update parameters)

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

---

Initialize $\theta$ using $\lambda_0$, $\lambda \leftarrow \lambda_0$

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

Step 1: (Lexical generation)

a. Set $\lambda_0 \leftarrow GENLEX(x, y; \lambda, \theta)$

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

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

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

Step 2: (Update parameters)

Output: Parameters $\theta$ and lexicon $\lambda$
Initialize θ using \( \Lambda_0 \) : \( \Lambda \leftarrow \Lambda_0 \)
For \( t = 1 \ldots T ; i = 1 \ldots n \) :

**Step 1:** (Lexical generation)
\[ a. \leftarrow \, \text{GEN} \]

\[ b. \text{LEX} \]

\[ c. \text{Select lexical entries} \]

\[ d. \text{Update lexicon} \]

**Step 2:** (Update parameters)

Output: Parameters \( \theta \) and lexicon \( \Lambda \)

---

Initialize θ using \( \Lambda_0 \) : \( \Lambda \leftarrow \Lambda_0 \)
For \( t = 1 \ldots T ; i = 1 \ldots n \) :

**Step 1:** (Lexical generation)
\[ a. \leftarrow \, \text{GEN} \]

\[ b. \text{LEX} \]

\[ c. \text{Select lexical entries} \]

\[ d. \text{Update lexicon} \]

**Step 2:** (Update parameters)

Output: Parameters \( \theta \) and lexicon \( \Lambda \)

---

Initialize θ using \( \Lambda_0 \) : \( \Lambda \leftarrow \Lambda_0 \)
For \( t = 1 \ldots T ; i = 1 \ldots n \) :

**Step 1:** (Lexical generation)
\[ a. \leftarrow \, \text{GEN} \]

\[ b. \text{LEX} \]

\[ c. \text{Select lexical entries} \]

\[ d. \text{Update lexicon} \]

**Step 2:** (Update parameters)

Output: Parameters \( \theta \) and lexicon \( \Lambda \)

---

Initialize θ using \( \Lambda_0 \) : \( \Lambda \leftarrow \Lambda_0 \)
For \( t = 1 \ldots T ; i = 1 \ldots n \) :

**Step 1:** (Lexical generation)
\[ a. \leftarrow \, \text{GEN} \]

\[ b. \text{LEX} \]

\[ c. \text{Select lexical entries} \]

\[ d. \text{Update lexicon} \]

**Step 2:** (Update parameters)

Output: Parameters \( \theta \) and lexicon \( \Lambda \)

---

Initialize θ using \( \Lambda_0 \) : \( \Lambda \leftarrow \Lambda_0 \)
For \( t = 1 \ldots T ; i = 1 \ldots n \) :

**Step 1:** (Lexical generation)
\[ a. \leftarrow \, \text{GEN} \]

\[ b. \text{LEX} \]

\[ c. \text{Select lexical entries} \]

\[ d. \text{Update lexicon} \]

**Step 2:** (Update parameters)

Output: Parameters \( \theta \) and lexicon \( \Lambda \)

---

Initialize θ using \( \Lambda_0 \) : \( \Lambda \leftarrow \Lambda_0 \)
For \( t = 1 \ldots T ; i = 1 \ldots n \) :

**Step 1:** (Lexical generation)
\[ a. \leftarrow \, \text{GEN} \]

\[ b. \text{LEX} \]

\[ c. \text{Select lexical entries} \]

\[ d. \text{Update lexicon} \]

**Step 2:** (Update parameters)

Output: Parameters \( \theta \) and lexicon \( \Lambda \)
Initialize $\theta$ using $A_0$, $\Lambda \leftarrow A_0$

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

**Step 1**: (Lexical generation)

**Step 2**: (Update parameters)

a. Set $G_i \leftarrow \text{MAXV}(\text{GEN}(x_i; \Lambda; \theta))$ and $B_i \leftarrow \{e \in \text{GEN}(x_i; \Lambda) : \neg \gamma_{\Lambda}(y, b)\}$

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

$$R_i \{g \mid g \in G_i \land \exists b \in B_i \text{ s.t. } \theta \Phi_i(g) - \Phi_i(b) < \gamma_{\Lambda}(g, b)\}$$

$$E_i \{b \mid b \in B_i \land \exists g \in G_i \text{ s.t. } \theta \Phi_i(g) - \Phi_i(b) < \gamma_{\Lambda}(g, b)\}$$

c. Apply the additive update:

$$\theta \leftarrow \theta + \frac{1}{T} \sum_{t=0}^{T} \theta \Phi_i(r)$$

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

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

Supervised

- \( \mathcal{V} \)

Template-based GENLEX

Unification-based GENLEX

Supervised Validation Function

- Validate logical form against gold label

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

y parse

\( z_i \) labeled logical form

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

Supervised Template-based GENLEX

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

I want a flight to new york

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

Supervised Template-based GENLEX

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

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

Need lexemes to instantiate templates

(Zettlemoyer and Collins 2005)
Supervised Template-based

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

I want a flight to new york
\\\(\lambda x. \text{flight}(x) \land \text{to}(x, \text{NYC})\)

All possible sub-strings

I want a flight
flight
flight to new

... 

Create lexemes

I want a flight
flight
flight to new

... 

Fast Parsing with Pruning

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

I want a flight from New York to Boston on Delta
\\(\lambda x. \text{from}(x, \text{NYC}) \land \text{to}(x, \text{BOS}) \land \text{carrier}(x, \text{DL})\)
Fast Parsing with Pruning

I want a flight from New York to Boston on Delta
\( \lambda x. \text{from}(x, \text{NYC}) \land \text{to}(x, \text{BOS}) \land \text{carrier}(x, \text{DL}) \)

Supervised Template-based GENLEX

Summary

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

Unification-based GENLEX

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

Unification-based GENLEX

- Initialize lexicon with labeled logical forms
  For every labeled training example:
  \( \lambda x. \text{flight}(x) \land \text{to}(x, \text{BOS}) \)
  Initialize the lexicon with:
  I want a flight to Boston \( \vdash S : \lambda x. \text{flight}(x) \land \text{to}(x, \text{BOS}) \)

Unification-based GENLEX

- Splitting lexical entries
  I want a flight to Boston \( \vdash S : \lambda x. \text{flight}(x) \land \text{to}(x, \text{BOS}) \)
  I want a flight \( \vdash S/(S|NP) : \lambda f. \lambda x. \text{flight}(x) \land f(x) \)
  to Boston \( \vdash S|NP : \lambda x. \text{to}(x, \text{BOS}) \)

Unification-based GENLEX

- Splitting CCG categories:
  1. Split logical form \( h \) to \( f \) and \( g \) s.t.
     \( f(g) = h \) or \( \lambda x. f(g(x)) = h \)
  2. Infer syntax from logical form type
     \( S/(S|NP) : \lambda f. \lambda x. \text{flight}(x) \land f(x) \)
     \( S|NP : \lambda x. \text{to}(x, \text{BOS}) \)
     \( S : \lambda x. \text{flight}(x) \land \text{to}(x, \text{BOS}) \)
     \( S/|NP : \lambda y. \lambda x. \text{flight}(x) \land f(x,y) \)
     \( |NP : \lambda x. \text{to}(x, \text{BOS}) \)
     ...
**Unification-based GENLEX**

- Split text and create all pairs

I want a flight to Boston

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

Parameter Initialization

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

I want a flight to Boston

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

Initial score for new lexical entries: average over pairwise weights

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

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

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

Iteration 2

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

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

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

Iteration 2

Experiments

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

Results

![Bar chart showing results for Geo880, ATIS, Geo250 English, Geo250 Spanish, Geo250 Japanese, Geo250 Turkish. The chart compares template-based, unification-based, and unification-based + factored lexicon methods.](chart.png)
**GENLEX Comparison**

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

**Recap CCGs**

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

[Steedman 1996, 2000]

**Unified Learning Algorithm**

Initialize \( \theta \) using \( \lambda_0 \), \( \Lambda \leftarrow \Lambda_0 \)

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

1. (Lexical generation)
2. (Update parameters)

Output: Parameters \( \theta \) and lexicon \( \Lambda \)

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

**Lexical Generation Procedure**

**Validation Function**

\( \mathcal{V} : \mathcal{Y} \rightarrow \{ t, f \} \)

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

**Learning Choices**

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

**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

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

\( y \in \mathcal{Y} \) parse
\( e_i \in \mathcal{E} \) available execution result
\( \text{EXEC}(y) : \mathcal{Y} \to \mathcal{E} \)

logical form at the root of \( y \)

[Artzi and Zettlemoyer 2013b]

Weakly Supervised Validation Function

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

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

Weakly Supervised Validation Function

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

Robot moving in an environment

Complete Demonstration
Validate all steps

Weakly Supervised Validation Function

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

I want a flight to new york

\( \text{flight}(x) \mapsto \{x, \text{NYC}\} \)

No access to labeled logical form

Initialize templates

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

I want \( \vdash \text{S/VP/VP} : \lambda x. \text{flight}(x) \)

flight to new \( \vdash \text{PN/VP} : \lambda x. \text{flight}(x) \)

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

I want a flight to new york

Use all logical constants in the system instead

Many more lexemes

Initialize templates

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

Coarse Ontology

Generalize types

Merge identically typed constants
Weakly Supervised 
\textit{GENLEX}(x, V; \Lambda, \theta)

I want a flight to new york

Create lexemes

flight

flight

flight

flight to new

... 

Keep only lexical entries that participate in complete parses, which score higher than the current best valid parse by a margin
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 \( \text{GENLEX}(x, V; \Lambda, \theta) \)

Results

Unified Learning Algorithm

Extensions

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

Modeling

Show me all papers about semantic parsing

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

What should these logical forms look like?

But why should we care?
Modeling Considerations

- Capture language complexity
- Satisfy system requirements
- Align with language units of meaning

Modeling is key to learning compact lexicons and high performing models.

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

Querying Databases

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

Noun Phrases

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

Verbs

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

Nouns

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

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

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

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

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

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

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

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

\[ \text{in}(x, \text{CO}) \]

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

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

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

Function Words

Certain words are used to modify syntactic roles

\[ \text{state that borders California} \]

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

\[ \{ \text{OR}, \text{NV}, \text{AZ} \} \]

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

Function Words

Definite Determiners

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

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

the mountain in Washington

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

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

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

\[
\text{NP} = \lambda x. f(x) \\
\text{N} = \lambda x. \text{state}(x) \\
\text{A} = \lambda x. \text{mountain}(x) \\
\text{x} = \lambda y. (\text{state}(y) \land \text{mountain}(y))
\]

**Indefinite Determiners**

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

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

A state with a mountain:

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

Exists

[Goodman 2011; Artzi and Zettlemoyer 2013]

**Superlatives**

Superlatives select optimal entities according to a measure.

the largest state

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

Min or max over this set according to this measure

WA, CA, ...
Superlatives

\[
\begin{align*}
\text{the largest state} & \quad \text{state} \\
\frac{\argmax_{x} \lambda x. f(x), y. \text{pop}(y))}{\lambda x. \text{state}(x)} \\
\frac{\text{np/n}}{\text{np}} \\
\text{represent the queries that generate their answers} \\
\text{reflects the query SQL}
\end{align*}
\]

Representing Questions

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

DB Queries

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

the blue triangle and the green arch

Plurality

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

Plurality

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

Plurality

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

Cognitive evidence for sets being a primitive type
[Sconyers et al. 2012]

Plurality

Plurality is a modifier and entities are defined to be sets.
arch
\[ \lambda x.\text{arch}(x) \land \text{sg}(x) \]
\[ \{\text{ }, \} \}
\[ \{\text{ }, \} \]
\[ \{\text{ }, \} \]
Plurality

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

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

Arches

\[
\{\text{arch}, \text{arch}, \text{arch}, \ldots\}
\]

Plurality and Determiners

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

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

\[
\{\text{arch}, \text{arch}, \text{arch}, \ldots\}
\]

Adjectives

Adjectives are conjunctive modifiers

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

Blue objects

\[
\{\text{blue object}, \text{blue object}, \ldots\}
\]

DBs and Physical Objects

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

Borders

State 1 | State 2
--------|--------
WA      | OR     
WA      | ID     
CA      | OR     
CA      | NV     
CA      | AZ

Beyond Queries

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

\[
\text{go forward along the stone hall to the intersection with a bare concrete hall}
\]

\[
\text{Verify(front: GRAVEL_HALL)}
\]

\[
\text{Travel()}
\]

\[
\text{Verify(side: CONCRETE_HALL)}
\]

[Chen and Mooney 2011]
Common approach to represent instructional language
- Natural for executing commands

Procedural Representations

leave the room and go right
do_seq(verify(room(current_loc)),
move_to(unique_thing(\(x\.equals(distance(x, 1))\))),
move_to(right_loc))

[Matuszek et al. 2012b]

Click Start, point to Search, and then 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")

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

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

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

Agent

Executing instructions

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

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

Grounded Resolution of Determiners

Nouns denote sets of objects

class

\lambda x. chair(x)

\{\}

Instructional Environment
Grounded Resolution of Determiners

Definite determiner selects a single entity
the chair $ix.chair(x)$

Fail?

Definite determiner depends on agent state

Agent

Definite determiner depends on agent state

Must disambiguate to select a single entity

Fail?

Definite determiner selects a single entity
the chair $ix.chair(x)$

Definite determiner selects a single entity
the chair $ix.chair(x)$

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

Davidson 1969 (quoted in Maienborn et al. 2010)

Davidson 1967

**Active**

Vincent shot Marvin

\[ \exists a. \text{shot}(a, \text{VINCENT}, \text{MARVIN}) \land \text{in}(a, x. \text{car}(x)) \land \neg \text{intentional}(a) \]

**Passive**

Marvin was shot (by Vincent)

Neo-Davidsonian Event Semantics

Agent optional in passive

Parsons 1990

[Parsons 1990]
Neo-Davidsonian Event Semantics

**Active**
Vincent shot Marvin

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

**Passive**
Marvin was shot (by Vincent)

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

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

**Active**
Vincent shot Marvin

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

**Passive**

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

[Parsons 1990]

---

Neo-Davidsonian Event Semantics

Vincent shot Marvin in the car accidentally

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

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

[Parsons 1990]

---

Representing Imperatives

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

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

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

**Modeling Instructions**

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

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

Disambiguate by preferring shorter sequences

**Modeling Instructions**

- Events can be modified by adverbials

\[
\text{move twice } \lambda a.\text{move}(a) \land \text{len}(a, 2)
\]

**Modeling Instructions**

- Events can be modified by adverbials

\[
\text{go to the chair } \lambda a.\text{move}(a) \land \text{to}(a, \text{cabinet})
\]

**Modeling Instructions**

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

World model changes during execution
move until you reach the chair
\( \lambda a. \text{move}(a) \land \text{post}(a, \text{intersect}(x, \text{chair}(x), \text{you})) \)

Never intersects

Update model to reflect state change

Implicit Actions

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

Implicit actions

Experiments

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

Demonstration:

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

Results

SAIL Corpus - Cross Validation

More Reading about Modeling

Type-Logical Semantics
by Bob Carpenter
Today

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

Looking Forward

- Challenges: Focus, consists of 86 domains, an average of 25
- Freebase (Bollacker et al., 2008) is a free, weight to make a difference during parsing. Prunes only a few lexical entries have high enough
- Tries with their predicted weights, although typically only a few lexical entries have high enough
- Average weight of all lexical entries in UBL with

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

Looking Forward: Code

- Goal: Program using natural language
- Challenges: - Data  - Complex intent  - Complex output
- See: Kushman and Barzilay 2013; Lei et al. 2013

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

Looking Forward: Sensors

- Goal: Integrate semantic parsing with rich sensing on real robots
- Challenges: - Data  - Managing uncertainty  - Interactive learning
Includes ready-to-run examples

[Artzi and Zettlemoyer 2013a]

Supplementary Material

Function Composition

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