Learning to Map Sentences to Logical Form

Luke Zettlemoyer

joint work with Michael Collins
MIT Computer Science and Artificial Intelligence Lab

A Challenging Problem

Building the mapping $M$, in the most general form, requires solving natural language understanding.

There are restricted domains that are still challenging:

- Natural language interfaces to databases
- Dialogue systems

Mapping Text to Meaning

Input: (text strings)

- Natural language text

Output: (formal meaning representation)

- A representation of the underlying meaning of the input text

Computation: (an algorithm $M$)

- Recovers the meaning of the input text

Learning The Mapping

Why learn:

- Difficult to build by hand
- Learned solutions are potentially more robust

We consider a supervised learning problem:

- Given a training set: $\{(NL_i, MR_i) \mid i=1...n\}$
- Find the mapping $M$ that best fits the training set
- Evaluate on unseen test set
The Setup for This Talk

**NL:** A single sentence
- usually a question

**MR:** A lambda-calculus expression
- similar to meaning representations used in formal semantics classes in linguistics

**M:** Weighted combinatory categorical grammar (CCG)
- mildly context-sensitive formalism
- explains a wide range of linguistic phenomena: coordination, long distance dependencies, etc.
- models syntax and semantics
- statistical parsing algorithms exist

The Setup for This Talk

A Simple Training Example

Given training examples like:

**Input:** What states border Texas?
**Output:** \( \lambda x. \text{state}(x) \land \text{borders}(x, \text{texas}) \)

**MR:** Lambda calculus
- Can think of as first-order logic with functions
- Useful for defining the semantics of questions

Challenge for learning:
- Derivations (parses) are not in training set
- We need to recover this missing information

More Training Examples

**Input:** What is the largest state?
**Output:** \( \text{argmax}(\lambda x. \text{state}(x), \lambda x. \text{size}(x)) \)

**Input:** What states border the largest state?
**Output:** \( \lambda x. \text{state}(x) \land \text{borders}(x, \text{argmax}(\lambda y. \text{state}(y), \lambda y. \text{size}(y))) \)

**Input:** What states border states that border states ... that border Texas?
**Output:** \( \lambda x. \text{state}(x) \land \exists y. \text{state}(y) \land \exists z. \text{state}(z) \land ... \land \text{borders}(x, y) \land \text{borders}(y, z) \land \text{borders}(z, \text{texas}) \)

Outline

- Combinatory Categorial Grammars (CCG)
- A learning algorithm: structure and parameters
- Extensions for spontaneous, unedited text
- Future Work: Context-dependent sentences
CCG Lexicon

Lexicon
- Pairs natural language phrases with syntactic and semantic information
- Relatively complex: contains almost all information used during parsing

Parsing Rules (Combinators)
- Small set of relatively simple rules
- Build parse trees bottom-up
- Construct syntax and semantics in parallel

<table>
<thead>
<tr>
<th>Words</th>
<th>Category</th>
<th>Syntax : Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texas</td>
<td>NP : texas</td>
<td></td>
</tr>
<tr>
<td>Kansas</td>
<td>NP : kansas</td>
<td></td>
</tr>
<tr>
<td>borders</td>
<td>(S\NP)/NP : λx.λy.borders(y,x)</td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>N : λx.state(x)</td>
<td></td>
</tr>
<tr>
<td>Kansas City</td>
<td>NP : kansas_city_MO</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Parsing: Lexical Lookup

<table>
<thead>
<tr>
<th>What</th>
<th>states</th>
<th>border</th>
<th>Texas</th>
</tr>
</thead>
<tbody>
<tr>
<td>S/(S\NP)/N</td>
<td>N</td>
<td>λ.x.λy.borders(y,x)</td>
<td>λx.state(x)</td>
</tr>
<tr>
<td>λ.f.λ.g.λx.f(x)λg(x)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Parsing Rules (Combinators)

Application
- X/Y : f  Y : a  =>  X : f(a)
- (S\NP)/NP  \( S \)  \( \lambda.x.\lambda.y.borders(y,x) \)  \( \lambda.y.borders(y,texas) \)  \( \lambda.x.state(x) \)  \( \lambda.x.state(x) \)
- Y : a  X/Y : f  =>  X : f(a)
- \( S \)  \( S \)  \( \lambda.y.borders(y,texas) \)  \( \lambda.y.borders(y,texas) \)  \( \lambda.y.borders(y,texas) \)  \( \lambda.y.borders(y,texas) \)
### Parsing a Question

<table>
<thead>
<tr>
<th>What</th>
<th>states</th>
<th>border</th>
<th>Texas</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S/(S\backslash NP) / N$</td>
<td>$N$</td>
<td>$\lambda y.\lambda x.\text{borders}(y, x)$</td>
<td>$\lambda x.\lambda y.\text{borders}(y, x)$</td>
</tr>
<tr>
<td>$\lambda f.\lambda g.\lambda x.\text{f}(x) \land g(x)$</td>
<td>$\lambda x.\text{state}(x)$</td>
<td>$S/(S\backslash NP)$</td>
<td>$S\backslash NP$</td>
</tr>
<tr>
<td>$\lambda g.\lambda x.\text{state}(x) \land g(x)$</td>
<td>$\lambda y.\text{borders}(y, x, \text{texas})$</td>
<td>$S/(S\backslash NP)$</td>
<td>$S\backslash NP$</td>
</tr>
<tr>
<td>$\lambda x.\text{state}(x) \land \text{borders}(x, \text{texas})$</td>
<td></td>
<td>$S$</td>
<td></td>
</tr>
</tbody>
</table>

### Parsing Rules (Combinators)

**Application**
- $X/Y : f \quad Y : a \Rightarrow X : f(a)$
- $Y : a \quad X\backslash Y : f \Rightarrow X : f(a)$

**Composition**
- $X/Y : f \quad Y/Z : g \Rightarrow X/Z : \lambda x. f(g(x))$
- $Y\backslash Z : g \quad X\backslash Y : f \Rightarrow X\backslash Z : \lambda x. f(g(x))$

**Other Combinators**
- Type Raising
- Crossed Composition

### Features, Weights and Scores

**Lexical count features:**

- $f(x,y) = [0, 1, 0, 1, 0, 0, 1, \ldots, 0]$
- $w = [-2, 0.1, 0, 2, 1, -3, 0, 0.3, \ldots, 0]$
- $w \cdot f(x,y) = 3.4$

### Weighted CCG

**Weighted linear model** $(\Lambda, f, w)$:
- **CCG lexicon**: $\Lambda$
- **Feature function**: $f(x,y) \in \mathbb{R}^m$
- **Weights**: $w \in \mathbb{R}^m$

**Quality of a parse $y$ for sentence $x$**
- **Score**: $w \cdot f(x,y)$
Weighted CCG Parsing

Two computations: sentence $x$, parses $y$, LF $z$

- Best parse
  
  $$ y^* = \underset{y}{\text{argmax}} \ w \cdot f(x, y) $$

- Best parse with logical form $z$
  
  $$ \hat{y} = \underset{y \text{ s.t. } L(y)=z}{\text{arg max}} \ w \cdot f(x, y) $$

We use a CKY-style dynamic-programming algorithm with pruning

A Supervised Learning Approach

Given a training set: $\{(x_i, z_i) \mid i=1\ldots n\}$

- $x_i$: a natural language sentence
- $z_i$: a lambda-calculus expression

Find a weighted CCG that minimizes error

- induce a lexicon $\Lambda$
- estimate weights $w$

Evaluate on unseen test set

Outline

- Combinatory Categorial Grammars (CCG)
- A learning algorithm: structure and parameters
- Extensions for spontaneous, unedited text
- Future Work: Context-dependent sentences

Learning: Two Parts

- GENLEX subprocedure
  
  - Create an overly general lexicon

- A full learning algorithm
  
  - Prunes the lexicon and estimates parameters $w$
Lexical Generation

**Input Training Example**

| Sentence: Texas borders Kansas | Logic Form: \( \text{borders(texas,kansas)} \) |

**Output Lexicon**

<table>
<thead>
<tr>
<th>Words</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texas</td>
<td>NP : texas</td>
</tr>
<tr>
<td>borders</td>
<td>((S\NP)/NP : \lambda x.\lambda y.\text{borders}(y,x))</td>
</tr>
<tr>
<td>Kansas</td>
<td>NP : kansas</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

---

**GENLEX**

- **Input**: a training example \((x_i, z_i)\)
- **Computation**:
  1. Create all substrings of words in \(x_i\)
  2. Create categories from logical form \(z_i\)
  3. Create lexical entries that are the cross product of these two sets
- **Output**: Lexicon \(\Lambda\)

---

**Step 1: GENLEX Words**

**Input Sentence:**

Texas borders Kansas

**Output Substrings:**

- Texas
- borders
- Kansas
- Texas borders
- borders Kansas
- Texas borders Kansas

---

**Step 2: GENLEX Categories**

**Input Logical Form:**

\( \text{borders(texas,kansas)} \)

**Output Categories:**

- ...
- ...
- ...
- ...
**Two GENLEX Rules**

<table>
<thead>
<tr>
<th>Input Trigger</th>
<th>Output Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>a constant $c$</td>
<td>NP : $c$</td>
</tr>
<tr>
<td>an arity two predicate $p$</td>
<td>$(S\setminus NP)/NP : \lambda x.\lambda y.p(y,x)$</td>
</tr>
</tbody>
</table>

Example Input: `borders(texas,kansas)`

Output Categories:
- NP : `texas`
- NP : `kansas`
- $(S\setminus NP)/NP : \lambda x.\lambda y.borders(y,x)`

**All of the Category Rules**

<table>
<thead>
<tr>
<th>Input Trigger</th>
<th>Output Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>a constant $c$</td>
<td>NP : $c$</td>
</tr>
<tr>
<td>arity one predicate $p$</td>
<td>N : $\lambda x.p(x)$</td>
</tr>
<tr>
<td>arity one predicate $p$</td>
<td>S/NP : $\lambda x.p(x)$</td>
</tr>
<tr>
<td>arity two predicate $p$</td>
<td>$(S\setminus NP)/NP : \lambda x.\lambda y.p(y,x)$</td>
</tr>
<tr>
<td>arity two predicate $p$</td>
<td>$(S\setminus NP)/NP : \lambda x.\lambda y.p(x,y)$</td>
</tr>
<tr>
<td>arity one function $f$</td>
<td>NP/N : $\lambda g.\lambda x.p(x)$</td>
</tr>
<tr>
<td>arity two predicate $p$</td>
<td>$(N\setminus N)/NP : \lambda x.\lambda y.\lambda z.p(x,y)$</td>
</tr>
<tr>
<td>arity one function $f$</td>
<td>S/NP : $\lambda x.f(x)$</td>
</tr>
</tbody>
</table>

**Step 3: GENLEX Cross Product**

**Input Training Example**

Sentence: `Texas borders Kansas`

Logic Form: `borders(texas,kansas)`

Output Lexicon

Output Substrings:
- Texas
- borders
- Kansas
- Texas borders
- borders Kansas

Output Categories:
- NP : `texas`
- NP : `kansas`
- $(S\setminus NP)/NP : \lambda x.\lambda y.borders(y,x)$

**GENLEX: Output Lexicon**

<table>
<thead>
<tr>
<th>Words</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texas</td>
<td>NP : <code>texas</code></td>
</tr>
<tr>
<td>Texas</td>
<td>NP : <code>kansas</code></td>
</tr>
<tr>
<td>Texas borders</td>
<td>$(S\setminus NP)/NP : \lambda x.\lambda y.borders(y,x)$</td>
</tr>
<tr>
<td>borders</td>
<td>NP : <code>texas</code></td>
</tr>
<tr>
<td>borders</td>
<td>NP : <code>kansas</code></td>
</tr>
<tr>
<td>$(S\setminus NP)/NP : \lambda x.\lambda y.borders(y,x)$</td>
<td>...</td>
</tr>
<tr>
<td>Texas borders</td>
<td>NP : <code>texas</code></td>
</tr>
<tr>
<td>Texas borders</td>
<td>NP : <code>kansas</code></td>
</tr>
<tr>
<td>Texas borders</td>
<td>$(S\setminus NP)/NP : \lambda x.\lambda y.borders(y,x)$</td>
</tr>
</tbody>
</table>

GENLEX is the cross product in these two output sets.
A Learning Algorithm

The approach is:
- **Online**: processes data set one example at a time
- **Able to Learn Structure**: selects a subset of the lexical entries from GENLEX
- **Error Driven**: uses perceptron-style parameter updates

### Inputs
- Training set \( \{(x_i, z_i) \mid i=1\ldots n \} \) of sentences and logical forms.
- Initial lexicon \( \Lambda \).
- Initial parameters \( w \).
- Number of iterations \( T \).

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

1. **Step 1: Check Correctness**
   - Let \( y^* = \arg\max_y w \cdot f(x_i, y) \)
   - If \( L(y^*) = z_i \), go to the next example

2. **Step 2: Lexical Generation**
   - Set \( \hat{y} = \arg\max_{y \in \Lambda} w \cdot f(x_i, y) \)
   - Define \( \lambda_i \) to be the lexical entries in \( \hat{y} \)
   - Set lexicon to \( \Lambda = \Lambda \cup \lambda_i \)

3. **Step 3: Update Parameters**
   - Let \( y' = \arg\max_y w \cdot f(x_i, y) \)
   - If \( L(y') \neq z_i \)
     - Set \( w = w + f(x_i, \hat{y}) - f(x_i, y') \)

### Output
Lexicon \( \Lambda \) and parameters \( w \).

---

Initialization

The initial lexicon has two types of entries:
- **Domain Independent**:
  - What | S/(S/NP)/N : \( \lambda f, \lambda g, \lambda x. f(x) \lambda g(x) \)
- **Domain Dependent**:
  - Texas | NP : texas

Initial features and weights
- **Features**: count the number of times each lexical entry is used in a parse
- **Initial Weights for Lexical Entries**:
  - From GENLEX: small negative values
  - From initial lexicon: small positive values

Related Work

Learning semantic parsers:
- Inductive Logic Prog.
- Machine Translation
- Probabilistic CFG Parsing
- Support Vector Mach.
- CCG:
- Log-linear models
- Multi-modal CCG
- Wide coverage semantics
- CCG Bank

[References]
- Zelle, Mooney 1996; Thompson, Mooney 2002
- Miller et. al. 1996; Ge, Mooney 2006
- Kate, Mooney 2006; Nguyen et al. 2006
- Steedman 1996, 2000
- Clark, Curran 2003
- Baldridge 2002
- Bos et al. 2004
- Hockenmaier 2003

Experimental Related Work

COCKTAIL: Tang and Mooney, 2001 (TM01)
- statistical shift-reduce parser learned with ILP techniques

λ-WASP: Wong and Mooney 2007 (WM07)
- Builds a synchronous CFG with statistical machine translation techniques

Experiments

Two database domains:
- Geo880: (geography)
  - 600 training examples
  - 280 test examples
- Jobs640: (job postings)
  - 500 training examples
  - 140 test examples

Evaluation

Test for completely correct semantics
- Precision:
  \[
  \frac{\text{# correct}}{\text{# total # parsed}}
  \]
- Recall:
  \[
  \frac{\text{# correct}}{\text{# total # sentences}}
  \]

Results

<table>
<thead>
<tr>
<th>Geo 880</th>
<th>Jobs 640</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prec.</td>
<td>Rec.</td>
</tr>
<tr>
<td>ZC05(^1)</td>
<td>96.25</td>
</tr>
<tr>
<td>WM07</td>
<td>93.71</td>
</tr>
<tr>
<td>TM01(^2)</td>
<td>89.92</td>
</tr>
</tbody>
</table>

\(^1\) Slightly different algorithm than just presented; performs similarly
\(^2\) Used 10-fold cross validation instead of the fixed test set
### Example Learned Lexical Entries

<table>
<thead>
<tr>
<th>Words</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>states</td>
<td>N : !x.state(x)</td>
</tr>
<tr>
<td>major</td>
<td>N/N : !g.λx.major(x) x.g(x)</td>
</tr>
<tr>
<td>population</td>
<td>N : !x.population(x)</td>
</tr>
<tr>
<td>cities</td>
<td>N : !x.city(x)</td>
</tr>
<tr>
<td>traverses</td>
<td>(S\NP)/NP : !x.λy.traverse(y,x)</td>
</tr>
<tr>
<td>run through</td>
<td>(S\NP)/NP : !x.λy.traverse(y,x)</td>
</tr>
<tr>
<td>the largest</td>
<td>NP/N : !g.argmax(g, !x.size(x))</td>
</tr>
<tr>
<td>rivers</td>
<td>N : !x.river(x)</td>
</tr>
<tr>
<td>the highest</td>
<td>NP/N : !g.argmax(g, !x.elev(x))</td>
</tr>
<tr>
<td>the longest</td>
<td>NP/N : !g.argmax(g, !x.len(x))</td>
</tr>
</tbody>
</table>

... ...

### Outline

- Combinatory Categorial Grammars (CCG)
- A learning algorithm: structure and parameters
- Extensions for spontaneous, unedited text
- Future Work: Context-dependent sentences

### A New Challenge

Learning CCG grammars works well for complex, grammatical sentences:

**Input:** Show me flights from Newark and New York to San Francisco or Oakland that are nonstop.

**Output:** !x.flight(x) ∧ nonstop(x) ∧ (from(x,NEW) ∨ from(x,NYC)) ∧ (to(x,SFO) ∨ to(x,OAK))

What about sentences that are common given spontaneous, unedited input?

**Input:** Boston to Prague the latest on Friday.

**Output:** argmax( !x.from(x,BOS) ∧ to(x,PRG) ∧ day(x,FRI) ∧ λy.time(y))

We will see an approach that works for both cases.

### Spontaneous, unedited input

The lexical entries that work for:

<table>
<thead>
<tr>
<th>Show me the latest flight from Boston to Prague on Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>S/NP</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Will not parse:

<table>
<thead>
<tr>
<th>Boston to Prague the latest on Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Relaxed Parsing Rules

Two changes:

• Add application and composition rules that relax word order
• Add type shifting rules to recover missing words

These rules significantly relax the grammar

• Introduce features to count the number of times each new rule is used in a parse
• Integrate into algorithm which should learn to penalize use

Review: Application

X/Y : f  Y : a  =>  X : f(a)
Y : a  X/Y : f  =>  X : f(a)

Disharmonic Application

• Reverse the direction of the principal category:

    X′/Y : f  Y : a  =>  X : f(a)
    Y : a  X/Y′ : f  =>  X : f(a)

flights          one_way
N
N/N
λ.x.flight(x)     λ.f.λ.x.f(x) ∧ one_way(x)

Review: Composition

X/Y : f  Y/Z : g  =>  X/Z : λ.x.f(g(x))
Y/Z : g  X/Y : f  =>  X/Z : λ.x.f(g(x))
Disharmonic Composition

- Reverse the direction of the principal category:

  \[ X \setminus Y : f \quad Y \setminus Z : g \implies X \setminus Z : \lambda x.f(g(x)) \]

  \[ Y \setminus Z : g \quad X \setminus Y : f \implies X \setminus Z : \lambda x.f(g(x)) \]

Missing content words

Insert missing semantic content

- NP : c \implies N \setminus N : \lambda f. \lambda x.f(x) \land p(x,c)

  flights
  Boston
  to Prague

  N
  \lambda x.flight(x)

  N
  BOS
  \lambda f. \lambda x.f(x) \land to(x,PRG)

  N
  \lambda x.f.argmax(\lambda x.f(x), \lambda x.time(x))

Missing content-free words

Bypass missing nouns

- N \setminus N : f \implies N : f(\lambda x.\text{true})

Northwest Air
  to Prague

  N\setminus N
  \lambda f. \lambda x.f(x) \land \text{airline}(x,NWA)

  N\setminus N
  \lambda f. \lambda x.f(x) \land to(x,PRG)

  N
  \lambda x.to(x,PRG)

  N
  \lambda x.airline(x,NWA) \land to(x,PRG)

A Complete Parse

Boston
  to Prague
  the latest
  on Friday

  NP
  BOS
  \lambda f. \lambda x.f(x) \land to(x,PRG)

  N\setminus N
  \lambda f. \lambda x.f(x) \land to(x,PRG)

  N\setminus N
  \lambda f. \lambda x.f(x) \land \text{day}(x,FRI)

  N
  \lambda x.day(x,FRI)

  N\setminus N
  \lambda f. \lambda x.f(x) \land \text{from}(x,BOS)

  N\setminus N
  \lambda f. \lambda x.f(x) \land \text{to}(x,PRG)

  N\setminus N
  \lambda x.day(x,FRI)

  N\setminus N
  \lambda f. \lambda x.f(x) \land \text{from}(x,BOS) \land \text{to}(x,PRG)

  N\setminus N
  \lambda x.day(x,FRI)

  N\setminus N
  \lambda f. \lambda x.f(x) \land \text{from}(x,BOS) \land \text{to}(x,PRG) \land \text{day}(x,FRI)

  N\setminus N
  \lambda x.day(x,FRI)
Inputs: Training set \{(x_i, z_i) \mid i=1...n\} of sentences and logical forms. Initial lexicon \Lambda. Initial parameters \(w\). Number of iterations \(T\).

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

Step 1: Check Correctness
- Let \(y^* = \text{argmax } w \cdot f(x_i, y)\)
- If \(L(y^*) = z_i\), go to the next example

Step 2: Lexical Generation
- Set \(\hat{y}_i = \text{GENLEX}(x_i, z_i)\)
- Let \(\hat{y} = \text{argmax } \{ y : L(y) = z_i \} \cdot w \cdot f(x_i, y)\)
- Define \(\lambda_i\) to be the lexical entries in \(\hat{y}\)
- Set lexicon to \(\Lambda = \Lambda \cup \lambda_i\)

Step 3: Update Parameters
- Let \(\hat{y} = \text{argmax } w \cdot f(x_i, y)\)
- If \(L(y) \neq z_i\)
  - Set \(w = w + f(x_i, \hat{y}) - f(x_i, y)\)

Output: Lexicon \(\Lambda\) and parameters \(w\).

Related Work for Evaluation

Hidden Vector State Model: He and Young 2006 (HY06)
- Learns a probabilistic push-down automaton with EM

\(\lambda\)-WASP: Wong & Mooney 2007 (WM07)
- Builds a synchronous CFG with statistical machine translation techniques

Zettlemoyer and Collins 2005 (ZC05)
- Uses GENLEX without relaxed grammar

Two Natural Language Interfaces

ATIS (travel planning)
- Manually-transcribed speech queries
- 4500 training examples
- 500 example development set
- 500 test examples

Geo880 (geography)
- Edited sentences
- 600 training examples
- 280 test examples

Evaluation Metrics

Precision, Recall, and F-measure for:
- Completely correct logical forms
- Attribute / value partial credit

\(\lambda x.\text{flight}(x) \land \text{from}(x, \text{BOS}) \land \text{to}(x, \text{PRG})\)

is represented as:

\(\{\text{flight, from = BOS, to = PRG}\}\)
Two-Pass Parsing

Simple method to improve recall:
• For each test sentence that can not be parsed:
  • Reparse with word skipping
  • Every skipped word adds a constant penalty
  • Output the highest scoring new parse
We report results with and without this two-pass parsing strategy

ATIS Test Set

Exact Match Accuracy:

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Pass</td>
<td>90.61</td>
<td>81.92</td>
<td>86.05</td>
</tr>
<tr>
<td>Two-Pass</td>
<td>85.75</td>
<td>84.60</td>
<td>85.16</td>
</tr>
</tbody>
</table>

ATIS Test Set

Partial Credit Accuracy:

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Pass</td>
<td>96.76</td>
<td>86.89</td>
<td>91.56</td>
</tr>
<tr>
<td>Two-Pass</td>
<td>95.11</td>
<td>96.71</td>
<td>95.9</td>
</tr>
<tr>
<td>HY 2006</td>
<td>---</td>
<td>---</td>
<td>90.3</td>
</tr>
</tbody>
</table>

Geo880 Test Set

Exact Match Accuracy:

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Pass</td>
<td>95.49</td>
<td>83.20</td>
<td>88.93</td>
</tr>
<tr>
<td>Two-Pass</td>
<td>91.63</td>
<td>86.07</td>
<td>88.76</td>
</tr>
<tr>
<td>ZC 05</td>
<td>96.25</td>
<td>79.29</td>
<td>86.95</td>
</tr>
<tr>
<td>WM 07</td>
<td>93.72</td>
<td>80.00</td>
<td>86.31</td>
</tr>
</tbody>
</table>
ATIS Development Set

Exact Match Accuracy:

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full method</td>
<td>87.26</td>
<td>74.44</td>
<td><strong>80.35</strong></td>
</tr>
<tr>
<td>Without features for new rules</td>
<td>70.33</td>
<td>42.45</td>
<td>52.95</td>
</tr>
<tr>
<td>Without relaxed word order rules</td>
<td>82.81</td>
<td>63.98</td>
<td>72.19</td>
</tr>
<tr>
<td>Without missing word rules</td>
<td>77.31</td>
<td>56.94</td>
<td>65.58</td>
</tr>
</tbody>
</table>

Summary

We presented an algorithm that:
- Learns the lexicon and parameters for a weighted CCG
- Uses online, error-driven updates

We extended it to parse spontaneous, unedited sentences
- Improves accuracy while maintaining the advantages of using a detailed grammatical formalism

We are currently working on learning context-dependent parsers

Future Work:
Meaning is context dependent

Input: Show me flights to Pittsburgh
Output: \( \lambda x. \text{flight}(x) \land \text{to}(x, \text{PIT}) \)

Input: from Boston nonstop
Output: \( \lambda x. \text{flight}(x) \land \text{nonstop}(x) \land \text{to}(x, \text{PIT}) \land \text{from}(x, \text{BOS}) \)

Input: Give me the cheapest one
Output: \( \text{argmin}(\lambda x. \text{flight}(x) \land \text{nonstop}(x) \land \text{to}(x, \text{PIT}) \land \text{from}(x, \text{BOS}), \lambda x. \text{cost}(x)) \)

Context-dependent data

Modified ATIS dialogues
- Extract user statements
- Label each statement with context-dependent meaning (by converting original SQL)

400 dialogues (=3000 queries)
- average 7.5 per dialogue, min 2, max 55
- All of the challenges from previous work still apply but must also model context
Can correct previous statements

Input: Show me flights to Pittsburgh on thursday night
Output: \( \lambda x. \text{flight}(x) \land \text{to}(x, \text{PIT}) \land \text{day}(x, \text{THU}) \land \text{during}(x, \text{PM}) \)

Input: friday before 10am
Output: \( \lambda x. \text{flight}(x) \land \text{to}(x, \text{PIT}) \land \text{day}(x, \text{FRI}) \land \text{time}(x) < 1000 \)

When do we copy content from previous statements, what do we change?

Can refer to sets

Input: What airlines fly to Pittsburgh.
Output: \( \lambda x. \text{airline}(x) \land \exists y. \text{flight}(y) \land \text{to}(y, \text{PIT}) \land \text{operates}(y, x) \)

Input: Which of these flights are nonstop.
Output: \( \lambda x. \text{flight}(x) \land \text{to}(x, \text{PIT}) \land \neg \text{nonstop}(x) \)

Which variables define the sets?

We may need world knowledge

Input: Show me flights from Boston to Pittsburgh
Output: \( \lambda x. \text{flight}(x) \land \text{from}(x, \text{BOS}) \land \text{to}(y, \text{PIT}) \)

Input: List return flights
Output: \( \lambda x. \text{flight}(x) \land \text{from}(x, \text{PIT}) \land \text{to}(y, \text{BOS}) \)

These types of sequences are challenging but relatively rare.