Executable Semantic Parsing

ESSLLI

Class 2

Instructor: Jonathan Berant
Class 1 reminder

We learned to manually write down logical forms for various natural language constructs

- Entities
- Relations
- Superlatives
- Events
- …

But how do we build logical forms automatically?
Outline

• Compositionality
  – Grammars and derivations
  – Example
  – Lexicon
  – Modeling
  – CCG
  – Exercise
Outline

• Compositionality
  – Grammars and derivations
  – Example
  – Lexicon
  – Modeling
  – CCG
  – Exercise
## Grammars and derivations

### Simple grammar

<table>
<thead>
<tr>
<th>Rule Type</th>
<th>Rule</th>
<th>Lexicon</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>(lexicon)</td>
<td>Set  ⇒ Phrase</td>
<td>Chicago  ⇒ Chicago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(lexicon)</td>
<td>Binary ⇒ Phrase</td>
<td>PlacesLived ⇒ lived in</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(join)</td>
<td>Set  ⇒ Binary Set</td>
<td>b.s  ⇒ b s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(intersect)</td>
<td>Root ⇒ Set Set</td>
<td>s₁ ⊓ s₂ ⇒ s₁ s₂</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Grammars and derivations

Simple grammar

(lexicon) Set $\Rightarrow$ Phrase

Chicago $\Rightarrow$ Chicago

(lexicon) Binary $\Rightarrow$ Phrase

PlacesLived $\Rightarrow$ lived in

(join) Set $\Rightarrow$ Binary Set

b.s $\Rightarrow$ b s

(intersect) Root $\Rightarrow$ Set Set

$s_1 \cap s_2 \Rightarrow s_1 s_2$

Type.Person $\cap$ PlaceLived.Chicago

intersect

Type.Person who PlaceLived.Chicago

join

lexicon

lexicon

people

PlaceLived

Chicago

lived in Chicago

lexicon

lexicon
## Grammar

### Simple grammar

<table>
<thead>
<tr>
<th>(lexicon)</th>
<th>Set</th>
<th>⇒</th>
<th>Phrase</th>
<th>Chicago</th>
<th>⇒</th>
<th>Chicago</th>
</tr>
</thead>
<tbody>
<tr>
<td>(lexicon)</td>
<td>Binary</td>
<td>⇒</td>
<td>Phrase</td>
<td>PlacesLived</td>
<td>⇒</td>
<td>lived in</td>
</tr>
<tr>
<td>(join)</td>
<td>Set</td>
<td>⇒</td>
<td>Binary Set</td>
<td>b.s</td>
<td>⇒</td>
<td>b s</td>
</tr>
<tr>
<td>(intersect)</td>
<td>Root</td>
<td>⇒</td>
<td>Set Set</td>
<td>$s_1 \cap s_2$</td>
<td>⇒</td>
<td>$s_1 \ s_2$</td>
</tr>
</tbody>
</table>

A grammar $G$ is a 4-tuple:

- $\mathcal{V}$: *chicago, live in*
- $\mathcal{N}$: *Binary, Set, Root*
- $\text{Root} \in \mathcal{N}$: start symbol
- $\mathcal{R}$: grammar rules
Grammar rules

Simple grammar

(lexicon) Set $\Rightarrow$ Phrase
Chicago $\Rightarrow$ Chicago

(lexicon) Binary $\Rightarrow$ Phrase
PlacesLived $\Rightarrow$ lived in

(join) Set $\Rightarrow$ Binary Set
$b.s$ $\Rightarrow$ b s

(intersect) Root $\Rightarrow$ Set Set
$s_1 \cap s_2$ $\Rightarrow$ s_1 s_2

A rule $r \in R$ has:

\[
\begin{align*}
A & \in \mathcal{N} : \text{ left-hand-side (LHS) non-terminal} \\
\alpha & \in (\mathcal{N} \cup \mathcal{V})^+ \text{ right-hand-side} \\
f & \text{ semantic function for building derivations}
\end{align*}
\]

Semantic functions are the key component!
A derivation tree $d^A_{i:j}$ over a span $x = (w_i, \ldots, w_{j-1})$:

- Spans $(w_i, \ldots, w_{j-1})$
- Has category $d.c = A \in N$
- Has logical form $d.z$
- Has children that are derivations or terminals
Semantic functions

A function \( f : D^k \rightarrow 2^D \)

Example:

- \( r = A \rightarrow B \ C \ [f] \)
- \( D_{i:j}^A = f(d_{i:k}^B, d_{k:j}^C) \)

Lexicon function:

\[
\text{LEX} ( \text{lincoln} ) = \left\{ \begin{array}{c} \text{ENTITY: AbeLincoln} \quad \text{ENTITY: LincolnFilm} \\
\text{lincoln} \quad \text{lincoln} \quad \text{lincoln} \\
\end{array} \right\}
\]
Semantic functions

Join function:

\[
\text{JOIN} \left( \begin{array}{c}
\text{ENTITY: AbeLincoln} \\
\text{abraham lincoln}
\end{array} , \begin{array}{c}
\text{BINARY: PlaceOfBirthOf} \\
\text{born}
\end{array} \right) = \left\{ \begin{array}{c}
\text{ENTITY: AbeLincoln} \\
\text{abraham lincoln}
\end{array} , \begin{array}{c}
\text{BINARY: PlaceOfBirthOf} \\
\text{born}
\end{array} \right\}
\]

Merge function:

\[
\text{INTERSECT} \left( \begin{array}{c}
\text{SET: Type.City} \\
city
\end{array} , \begin{array}{c}
\text{SET: ReleaseDateOf.LincolnFilm} \\
\text{abraham lincoln born}
\end{array} \right) = \emptyset
\]

Return sets (unlike CFGs)
**Semantic function**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IDENTITYFn</code></td>
<td>Copy logical form</td>
</tr>
<tr>
<td><code>SELECTFn</code></td>
<td>Select one child logical form</td>
</tr>
<tr>
<td><code>DATEFn</code></td>
<td>Convert date descriptions to date</td>
</tr>
<tr>
<td><code>FILTERPOSFn</code></td>
<td>Filter words by POS</td>
</tr>
</tbody>
</table>
Semantic function

- **IDENTITYFn**: Copy logical form
- **SELECTFn**: Select one child logical form
- **DATEFn**: Convert date descriptions to date
- **FILTERPOSFn**: Filter words by POS
- **CONTEXTFn**: Integrate arbitrary context

*What city was abraham lincoln born?*
*How about his wife?*
Semantic function

**IdentityFn**  Copy logical form
**SelectFn**  Select one child logical form
**DateFn**  Convert date descriptions to date
**FilterPosFn**  Filter words by POS
**ContextFn**  Integrate arbitrary context

*What city was abraham lincoln born?*

*How about his wife?*

**Flexible! arbitrary logic can be used**

\[ \lambda x. \text{Border}(x, \text{Italy}) \]

\[ \lambda y. \lambda x. \text{Border}(x, y) \quad \text{ITALY} \]

\[ S \setminus N \]

\[ \Rightarrow \quad (S \setminus N)/N \quad \text{N} \]

bordering Italy

bordering Italy
Outline

• Compositionality
  – Grammars and derivations
  – **Example**
  – Lexicon
  – Modeling
  – CCG
  – Exercise
Example

./run @mode=simple-lambdadcs

-Grammar.inPaths esslli_2016/class2_demo.grammar

-SimpleLexicon.inPaths esslli_2016/class2_demo.lexicon
Example

california
the golden state
cities in the golden state
towns located in california
rivers bordering california
Example

california
the golden state
cities in the golden state
towns located in california
rivers bordering california

Is the last utterance meaningful?

How can we avoid such logical forms?
Outline

• Compositionality
  – Grammars and derivations
  – Example
  – **Lexicon**
  – Modeling
  – CCG
  – Exercise
The lexicon problem

- california
- the golden state
- cities
- towns
- in

How is the lexicon generated?
The lexicon problem

california
the golden state
cities
towns
in

California
California
Type.Citytown
Type.CityTown
ContainedBy

How is the lexicon generated?

- Annotation
- Exhaustive search
- String matching
- Supervised alignment
- Unsupervised alignment
- Learning
Annotation

where  Type.Location
most  \( \lambda x.\lambda y.\text{ARGMAX}(y, x) \)
how many  \( \lambda x.\text{COUNT}(x) \)

Useful for function words
Annotation

where Type.Location
most $\lambda x.\lambda y.\text{ARGMAX}(y, x)$
how many $\lambda x.\text{COUNT}(x)$

Useful for function words

city Type.CityTown
lake Type.Lake
area Area

Useful for small domains
Annotation

where \( \text{Type.Location} \)

most \( \lambda x.\lambda y.\text{ARGMAX}(y, x) \)

how many \( \lambda x.\text{COUNT}(x) \)

Useful for function words

city \( \text{Type.CityTown} \)
lake \( \text{Type.Lake} \)
area \( \text{Area} \)

Useful for small domains

Large domains? Different languages?
Exhaustive search

city | Type.Lake
state | Population
people | Area
area | $\lambda x.\text{COUNT}(x)$
how many | Type.CityTown
… | …
Exhaustive search

Size of lexicon: $|\mathcal{V}| \times |\mathcal{L}|$

Intractable for large domains
Exhaustive search

\[
\begin{aligned}
\text{city} & \quad \text{Type.Lake} \\
\text{state} & \quad \text{Population} \\
\text{people} & \quad \text{Area} \\
\text{area} & \quad \lambda x. \text{COUNT}(x) \\
\text{how many} & \quad \text{Type.CityTown} \\
\ldots & \quad \ldots
\end{aligned}
\]

Size of lexicon: \(|V| \times |L|

Intractable for large domains

Exhaustive?

\[
\begin{aligned}
\text{actress} & \quad \text{PROFESSION.Actor} \sqsubseteq \text{GENDER.FEMALE} \\
\text{relative} & \quad \lambda x. \text{PARENT}.x \sqsubseteq \text{SIBLING}.x \sqsubseteq \text{CHILD}.x \\
\text{expat} & \quad \mu x. \text{BirthPlaceOf}.x \neq \text{ARGMAX}(x, \text{PlacesLived})
\end{aligned}
\]
String match

Knowledge-bases often have language descriptions

- Language specific
- Limited coverage and noisy
String match

Knowledge-bases often have language descriptions

- Language specific
- Limited coverage and noisy
- Useful as a feature at training time
Supervised alignment

Assume access to \((x, z)\) language-logical form pairs

*State with the largest area*

\[
\text{argmax}(\text{Type.State}, \text{Area})
\]
Supervised alignment

Assume access to \((x, z)\) language-logical form pairs

*State with the largest area*

\[ \text{argmax}(\text{Type.State, Area}) \]

Define patterns over \((x, z)\) to generate candidate lexicon entries

- \(\text{State with the largest area} \)
- \(\lambda x. \text{Type}.x\)
- \(\lambda x. \lambda y. \text{ARGMAX}(x, y)\)
Supervised alignment

Assume access to \((x, z)\) language-logical form pairs

\textit{State with the largest area}

\texttt{argmax}(Type(State, Area))

Define patterns over \((x, z)\) to generate candidate lexicon entries

\begin{align*}
\text{State} & \quad \text{State} \\
\text{with the largest area} & \quad \text{Type.State} \\
\text{largest area} & \quad \lambda x. \text{Type}.x \\
& \quad \text{Area} \\
& \quad \lambda x. \lambda y. \text{ARGMAX}(x, y)
\end{align*}

Keep counts for each lexical entry as a feature
Supervised alignment

Assume access to \((x, z)\) language-logical form pairs

State with the largest area
\[
\text{argmax}(\text{Type.State, Area})
\]

Define patterns over \((x, z)\) to generate candidate lexicon entries

\[
\begin{align*}
\text{State} & \\
\text{with the largest} & \\
\text{largest} & \\
\text{area} & \end{align*}
\begin{align*}
\text{State} & \\
\text{Type.State} & \\
\lambda x.\text{Type}.x & \\
\text{Area} & \\
\lambda x.\lambda y.\text{ARGMAX}(x, y) &
\end{align*}
\]

Keep counts for each lexical entry as a feature

Predefined patterns

Generalization?
Unsupervised alignment

(Barack Obama, was born in, Honolulu)
(Albert Einstein, was born in, Ulm)
(Barack Obama, lived in, Chicago)

... 15M triples ...

(Barack Obama, HasBirthplace, Honolulu)
(Albert Einstein, HasBirthplace, Ulm)
(Barack Obama, PlacesLived.Location, Chicago)

... 600M triples ...
Unsupervised alignment

grew up in [Person, Location]
born in [Person, Date]
marrried in [Person, Date]
born in [Person, Location]

DateOfBirth
PlaceOfBirth
Marriage.StartDate
PlacesLived.Location

(RandomPerson, Seattle)
(BarackObama, Honolulu)
(MichelleObama, Chicago)
(BarackObama, Chicago)
Unsupervised alignment

grew up in [Person, Location]
born in [Person, Date]
mari**e**d in [Person, Date]
born in [Person, Location]

Alignment features
phrase-count: 15,765
intersection-count: 6,048

(RandomPerson, Seattle) (BarackObama, Honolulu) (MichelleObama, Chicago)
(BarackObama, Chicago)
Unsupervised alignment

grew up in [Person, Location]  
born in [Person, Date]  
made in [Person, Date]  
born in [Person, Location]

Alignment features
phrase-count: 15,765
intersection-count: 6,048

Generalizes to test set
Restricted domain
Advanced

- Change lexicon at training time
- Batch voting
- Generative models

[Kwiatkowski et al., 2010; Kwiatkowski et al. 2011; Artzi et al., 2014; Krishnamurthy, 2015]
Outline

• Compositionality
  – Grammars and derivations
  – Example
  – Lexicon
  – Modeling
  – CCG
  – Exercise
Bridging/trace predicates

Problem: Language defines semantics implicitly

Italy’s language  Type.Language ⊓ LanguageOf.Italy
Where is Beijing  Type.Location ⊓ Contains.Beijing
Boston flight  Type.Flight ⊓ To.Boston
Bridging/trace predicates

Problem: Language defines semantics implicitly

Italy’s language \( \text{Type.Language} \sqcap \text{LanguageOf.Italy} \)
Where is Beijing \( \text{Type.Location} \sqcap \text{Contains.Beijing} \)
Boston flight \( \text{Type.Flight} \sqcap \text{To.Boston} \)

Lexicon?

’s \( \text{LanguageOf} \)

is \( \text{Contains} \)

??? \( \text{To} \)
Naive solution

language Type.Language
language LanguageOf
where Type.Location
where Contains
flight Type.Filght
flight To
flight From

More pressure on lexicon learning
Naive solution

```
language  Type.Language
language  LanguageOf
where     Type.Location
where     Contains
flight    Type.Filght
flight    To
flight    From
```

More pressure on lexicon learning

Infer missing predicates from context

Can be implemented as a semantic function! (BridgeFn)
Bridging 1: two unaries

Boston

Type.Flight

boston

lex

flight

lex
Bridging 1: two unaries

Type.Flight ⊓ To.Boston

Insert any type-matching binary

(bridge first) \[ \text{Set} \implies \text{Set Set} \]
Bridging 1: two unaries

Type.Flight ⊓ To.Boston

Insert any type-matching binary

(bridge first) \[\text{Set} \Rightarrow \text{Set Set}\]

Learn to choose implicit predicate
Bridging 2: single unary

Where is Beijing

Beijing

lex

Where is Beijing
Bridging 2: single unary

\begin{equation*}
\text{bridging}
\end{equation*}

Contains
\begin{equation*}
\mid
\mid
\mid
\text{Beijing}
\end{equation*}
\begin{equation*}
\text{lex}
\end{equation*}

\begin{equation*}
\text{Where } is \text{ Beijing}
\end{equation*}

Insert any type-matching binary

\begin{equation*}
\text{Contains.Beijing}
\end{equation*}

\begin{equation*}
\text{(bridge) } \text{Set } \Rightarrow \text{ Set}
\end{equation*}

More uncertainty
Bridging 3: n-ary relations

Marriage.Spouse.Madonna

join

Madonna

lex

Who did Madonna marry in 2000

join

Marriage.Spouse

lex

2000

Marry

in

2000
Bridging 3: n-ary relations

Marriage.Spouse.Madonna \join \join \textbf{bridging} \quad \text{Marriage.StartDate}

\textbf{lex} \quad \textbf{lex}

Who \quad did \quad Madonna \quad marry \quad in \quad 2000

Marriage.(Spouse.Madonna \sqcap \text{StartDate}.2000)
Bridging 3: n-ary relations

Marriage. Spouse. Madonna \join Marriage. Spouse \join Madagascar

Later: a more general solution called floating parser

- Wang et al., 2015; Pasupat and Liang, 2015
Type raising

Traverses Oregon

Traverse(Oregon)

Traverse  Oregon

traverses  Oregon
Type raising

Traverses Oregon

\[ \text{Traverse}(\text{Oregon}) \]

\[ \text{Traverse(\text{Oregon})} \]

\[ \text{Traverse(\text{Oregon})} \]

\[ \text{Traverse} \quad \text{Oregon} \]

\[ \text{traverses} \quad \text{Oregon} \]

traverses Oregon or traverses Nevada

\[ \text{Traverse(\text{Oregon})} \sqcup \text{Traverse(\text{Nevada})} \]

\[ \text{Traverse(\text{Oregon})} \quad \text{or} \quad \text{Traverse(\text{Nevada})} \]

\[ \text{Traverse(\text{Oregon})} \]

\[ \text{Traverse(\text{Nevada})} \]

\[ \text{Traverse} \quad \text{Oregon} \]

\[ \text{Traverse} \quad \text{Nevada} \]

\[ \text{traverses} \quad \text{Oregon} \]

\[ \text{traverses} \quad \text{Nevada} \]
Type raising

Traverse(Oregon) \sqcup Nevada

Traverse(Oregon) or Nevada

Traverse Oregon Nevada

traverses Oregon

How to apply the binary on multiple arguments?
Solution 1

\((\text{ConstFn } \lambda x.\lambda y.\lambda b. b(x) \uplus b(y)) \quad \text{Disj} \quad \Rightarrow \quad \text{or}\)
Solution 1

\[ \text{Disj} \implies \text{or} \]

\( (\text{ConstFn}\ \lambda x.\lambda y.\lambda b.\, b(x) \sqcup b(y)) \]

Traverses Oregon or Nevada or Colorado

Another lexical entry?
Type raising

Type raise the entity to a function that is applied on a binary

\[(\text{ApplyFn } \lambda x.\lambda f.f(x))\] \quad \text{FuncSet} \Rightarrow \text{Set}

\[(\text{ConstFn } \lambda a.\lambda b.\lambda g.a(g) \sqcup b(g))\] \quad \text{Disj} \Rightarrow or

Traverse(Oregon) \sqcup Traverse(Nevada)

\begin{align*}
\text{Traverse} & \quad \lambda g.g(\text{Nevada}) \sqcup g(\text{Oregon}) \\
\text{traverses} & \quad \lambda f.f(\text{Oregon}) \\
\text{Oregon} & \quad \lambda a.\lambda b.\lambda g.a(g) \sqcup b(g) \\
\text{Oregon} & \quad \lambda f.f(\text{Nevada}) \\
\text{or} & \quad \text{Nevada}
\end{align*}
Outline

- Compositionality
  - Grammars and derivations
  - Example
  - Lexicon
  - Modeling
  - CCG
  - Exercise
CCG

Standard formalism in semantics
Standard formalism in semantics

Lexicon:

\[
\begin{align*}
\text{fun} & \vdash \text{ADJ} : \lambda x. \text{fun}(x) \\
\text{fun} & \vdash \text{N/N} : \lambda f. \lambda x. \text{fun}(x) \land f(x)
\end{align*}
\]
CCG

Standard formalism in semantics

Lexicon:

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

\[
\text{fun} \vdash \text{N/N} : \lambda f. \lambda x. \text{fun}(x) \land f(x)
\]

Handful of combinators (grammar)

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

\[
B : g \quad A \backslash B : f \Rightarrow A : f(g) \quad (<)
\]
CCG

Main difference:

- CCG pushed complexity from grammar to lexicon
- lambda-DCS tries to push it from lexicon to learning
CCG

Main difference:

- CCG pushed complexity from grammar to lexicon
- lambda-DCS tries to push it from lexicon to learning

Rationale:

- CCG originates in linguistics
- Lambda-DCS originates in NLP
CCG

Main difference:

• CCG pushed complexity from grammar to lexicon
• lambda-DCS tries to push it from lexicon to learning

Rationale:

• CCG originates in linguistics
• Lambda-DCS originates in NLP
• CCG-based semantic parsing addressed many issues with type raising, factored lexicons, etc.  [Zettlemoyer and Collins, 2007; Kwiatkowski et al., 2011]
Philosophy: simple grammar and lexicon

- Capturing correct derivations only requires complex rules
Philosophy: simple grammar and lexicon

- Capturing correct derivations only requires complex rules
- Simple rules generate overapproximation of good derivations
Philosophy: simple grammar and lexicon

- Capturing correct derivations only requires complex rules
- Simple rules generate overapproximation of good derivations

Disambiguate at learning time!
Outline

• Compositionality
  – Grammars and derivations
  – Example
  – Lexicon
  – Modeling
  – CCG
  – Exercise
Exercise

Write a grammar that parses utterances

You can use class2_demo.lexicon and class2_demo.grammar that are on the website

They are also at /home/shared/esslii/semparse

You can also start with ConstantFn entries that parse the entire utterance and decompose
Exercise

State with the largest area
Top 5 cities by area
Major cities with at least 140,000 inhabitants
How many states have capitals with at least 200,000 inhabitants
State bordering Oregon and Nevada

Check if the grammar parses the sentence:

States bordering Oregon and Nevada and Arizona
Summary

• Grammars build derivations which compositionally provide logical forms

• Lexicons map phrases to basic logical forms

• Learning lexicons is hard so we prefer to push the complexity to learning