Recall: Dependencies

- Dependency syntax: syntactic structure is defined by dependencies
  - Head (parent, governor) connected to dependent (child, modifier)
  - Each word has exactly one parent except for the ROOT symbol
  - Dependencies must form a directed acyclic graph

Recall: Shift-Reduce Parsing

- I ate some spaghetti bolognese
- State: Stack: [ROOT I ate] Buffer: [some spaghetti bolognese]
- Left-arc (reduce operation): Let $\sigma$ denote the stack
  - “Pop two elements, add an arc, put them back on the stack”
  - $\sigma \downarrow w_{-2}, w_{-1} \rightarrow \sigma \uparrow w_{-1}$, $w_{-2}$ is now a child of $w_{-1}$
- State: Stack: [ROOT ate] Buffer: [some spaghetti bolognese]
Where are we now?

- Early in the class: sentences are just sequences of words
- Now we can understand them in terms of tree structures as well
- Why is this useful? What does this allow us to do?
- We’re going to see how parsing can be a stepping stone towards more formal representations of language meaning

Today

- First-order logic
- Compositional semantics with first-order logic
- CCG parsing for database queries
- Lambda-DCS for question answering

First-Order Logic

- Powerful logic formalism including things like entities, relations, and quantifications
- Propositions: let a = *It is day*, b = *It is night*
  - a ∨ b = either a is true or b is true, a => ¬b = a implies not b
- More complex statements: “Lady Gaga sings”
- Sings is a *predicate* (with one argument), function f: entity => true/false
- Sings(Lady Gaga) = true or false, have to execute this against some database (called a *world*)
- [[sings]] = *denotation*, set of entities which sing (sort of like executing this predicate on the world — we’ll come back to this)
Quantification

- Universal quantification: “for all” operator
  \( \forall x \text{sings}(x) \vee \text{dances}(x) \Rightarrow \text{performs}(x) \)
  “Everyone who sings or dances performs”

- Existential quantification: “there exists” operator
  \( \exists x \text{sings}(x) \)
  “Someone sings”

- Source of ambiguity! “Everyone is friends with someone”
  \( \forall x \exists y \text{friend}(x,y) \)

Logic in NLP

- Question answering:
  *Who are all the American singers named Amy?*

  \( \lambda x. \text{nationality}(x, \text{USA}) \land \text{sings}(x) \land \text{firstName}(x, \text{Amy}) \)

- Function that maps from x to true/false, like \text{filter}. Execute this on the world to answer the question

- Lambda calculus: powerful system for expressing these functions

- Information extraction: *Lady Gaga and Eminem are both musicians*
  \( \text{musician}(\text{Lady Gaga}) \land \text{musician}(\text{Eminem}) \)

- Can now do reasoning. Maybe know: \( \forall x \text{musician}(x) \Rightarrow \text{performer}(x) \)
  Then: \( \text{performer}(\text{Lady Gaga}) \land \text{performer}(\text{Eminem}) \)

Truth-Conditional Semantics

Database containing entities, predicates, etc.

- Truth-conditional semantics: sentence expresses something about the world which is either true or false

- Denotation: evaluation of some expression against this database

\( [[ \text{Lady Gaga} ]] = \text{e470} \)

\( [[ \text{sings(e470)} ]] = \text{True} \)

denotation of this string is an entity

denotation of this expression is T/F
Parses to Logical Forms

*Function application:* apply this to e470

\[
sings(e470)
\]

\[
te470 \rightarrow \text{NP} \rightarrow \text{VP} \rightarrow \lambda y. \text{sings}(y)
\]

\[
\lambda y. \text{sings}(y)
\]

takes one argument \((y, \text{the entity})\) and
returns a logical form \(\text{sings}(y)\)

- We can use the syntactic parse as a bridge to the lambda-calculus representation, build up a logical form *compositionally*

### General rules:
- **S:** \(f(x) \rightarrow \text{NP:** } x \text{ VP:** } f(x)\)
- **NP:** \(\exists e. \text{type}(e, \text{sing}) \land \text{agent}(e, e470) \land \text{manner}(e, \text{well}) \land \text{time}(e, ...)\)

### Tricky things

- **Adverbs/temporality:** *Lady Gaga sang well yesterday*
  \[
sings(\text{Lady Gaga, time=yesterday, manner=well})
\]

- **“Neo-Davidsonian” view of events:** things with many properties:
  \[
  \exists e. \text{type}(e, \text{sing}) \land \text{agent}(e, e470) \land \text{manner}(e, \text{well}) \land \text{time}(e, ...)
  \]

- **Quantification:** *Everyone is friends with someone*
  \[
  \exists x \forall y \text{ friend}(x, y) \quad \forall x \exists y \text{ friend}(x, y).
  \]

- **Same syntactic parse for both! So syntax doesn't resolve all ambiguities**

- **Indefinite:** *Amy ate a waffle*
  \[
  \exists w. \text{waffle}(w) \land \text{ate}(\text{Amy}, w)
  \]

- **Generic:** *Cats eat mice* (all cats eat mice? most cats? some cats?)
QA from Parsing

\[
\lambda x. \text{born}(e470, x)
\]

- Execute this function against a knowledge base to answer the question

- Tricky to parse due to wh-movement...would be easier if we said
  \[\text{Lady Gaga was born when}\]

Semantic Parsing

- For question answering, syntactic parsing doesn’t tell you everything you want to know, but indicates the right structure

- Solution: semantic parsing: many forms of this task depending on semantic formalisms

- Two today: CCG (looks like what we’ve been doing) and lambda-DCS

Combinatory Categorial Grammar

- Steedman+Szabolcsi 1980s: formalism bridging syntax and semantics

- Parallel derivations of syntactic parse and lambda calculus expression

- Syntactic categories (for this lecture): S, NP, “slash” categories

- S\NP: “if I combine with an NP on my left side, I form a sentence” — verb

- When you apply this, there has to be a parallel instance of function application on the semantics side
Combinatory Categorial Grammar

- Steedman+Szabolcsi 1980s: formalism bridging syntax and semantics
- Syntactic categories (for this lecture): S, NP, “slash” categories
  - S\NP: “if I combine with an NP on my left side, I form a sentence” — verb
  - (S\NP)/NP: “I need an NP on my right and then on my left” — verb with a direct object

Syntactic categories (for this lecture):
- S:
  - “if I combine with an NP on my left side, I form a sentence” — verb
- S\NP:
  - “I need an NP on my right and then on my left” — verb

CCG Parsing

- “What” is a very complex type: needs a noun and needs a S\NP to form a sentence. S\NP is basically a verb phrase (border Texas)
- Lexicon is highly ambiguous — all the challenge of CCG parsing is in picking the right lexicon entries

CCG Parsing

- Model: log-linear model over derivations with features on rules:
  \[ P(d|x) \propto \exp \mathbf{w}^T \left( \sum_{r \in d} f(r, x) \right) \]

Building CCG Parsers

- “to” needs an NP (destination) and N (parent)

Slide credit: Dan Klein

Zeilemoyer and Collins (2005)
Building CCG Parsers

- Training data looks like pairs of sentences and logical forms

What states border Texas ⊢ λx. state(x) ∧ borders(x, e89)

- Problem: we don’t know the derivation

  - Texas corresponds to NP | e89 in the logical form (easy to figure out)
  - What corresponds to (S/(S\NP))/N | λf.λg.λx. f(x) ∧ g(x)

  - How do we infer that without being told it?

Zettlemoyer and Collins (2005)

Lexicon

- GENLEX: takes sentence S and logical form L. Break up logical form into chunks C(L), assume any substring of S might map to any chunk

What states border Texas ⊢ λx. state(x) ∧ borders(x, e89)

- Chunks inferred from the logic form based on rules:

  - NP: e89 ⊢ (S\NP)/NP: λx. λy. borders(x, y)

  - Any substring can parse to any of these in the lexicon

    - Texas -> NP: e89 is correct
    - border Texas -> NP: e89
    - What states border Texas -> NP: e89

    ...

Zettlemoyer and Collins (2005)

GENLEX

<table>
<thead>
<tr>
<th>Input Trigger</th>
<th>Rule</th>
<th>Output Category</th>
<th>Category produced from logical form</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant c</td>
<td>NP</td>
<td>N</td>
<td>N \ λx. state(x) ∧ borders(x, e89)</td>
</tr>
<tr>
<td>any one predicate p₁</td>
<td>(S\NP)/NP: λx. p₁(x)</td>
<td>S\NP: λx. state(x)</td>
<td></td>
</tr>
<tr>
<td>any one predicate p₂</td>
<td>(S\NP)/NP: λx. p₂(x, e89)</td>
<td>(S\NP)/NP: λx. state(x)</td>
<td></td>
</tr>
<tr>
<td>any two predicate p₁</td>
<td>(S\NP)/NP: λx. λy. p₁(x, y)</td>
<td>(S\NP)/NP: λx. state(x, e89)</td>
<td></td>
</tr>
<tr>
<td>any two predicate p₂</td>
<td>(S\NP)/NP: λx. λy. p₂(x, y)</td>
<td>(S\NP)/NP: λx. state(x, e89)</td>
<td></td>
</tr>
<tr>
<td>literal with any two predicate p₁ and constant second argument c</td>
<td>N\NP: λx. λp₁(x, c) ∧ p(x)</td>
<td>N\NP: λx. λp₁(x, c) ∧ p(x)</td>
<td></td>
</tr>
<tr>
<td>any two predicate p₂</td>
<td>N\NP: λx. λp₂(x, y) ∧ p(x)</td>
<td>N\NP: λx. λp₂(x, y) ∧ p(x)</td>
<td></td>
</tr>
<tr>
<td>any one second argument of one function f</td>
<td>N\NP: λx. arg max / min(p, λx. f(x))</td>
<td>N\NP: λx. arg max / min(p, λx. f(x))</td>
<td></td>
</tr>
<tr>
<td>any one</td>
<td>S\NP: λx. f(x)</td>
<td>S\NP: λx. f(x)</td>
<td></td>
</tr>
</tbody>
</table>

- Very complex and hand-engineered way of taking lambda calculus expressions and “backsolving” for the derivation

Zettlemoyer and Collins (2005)

Learning

- Iterative procedure like the EM algorithm: estimate “best” parses that derive each logical form, retrain the parser using these parses with supervised learning

  - We’ll talk about a simpler form of this in a few slides
Applications

- GeoQuery: answering questions about states (~80% accuracy)
- Jobs: answering questions about job postings (~80% accuracy)
- ATIS: flight search
- Can do well on all of these tasks if you handcraft systems and use plenty of training data: these domains aren’t that rich
- What about broader QA?

Lambda-DCS

- Dependency-based compositional semantics — original version was less powerful than lambda calculus, lambda-DCS is as powerful
- Designed in the context of building a QA system from Freebase
- Freebase: set of entities and relations
  - March 15, 1961
  - Bob Cooper
  - Alice Smith
  - DateOfBirth
  - CapitalOf
  - PlaceOfBirth
  - PlaceOfBirth.Seattle
  - looks like a tree fragment over Freebase
  - \([\text{PlaceOfBirth}] = \text{set of pairs of (person, location)}\)
  - Liang et al. (2011), Liang (2013)
Lambda-DCS

March 15, 1961

Bob Cooper

Washington

Liang et al. (2011), Liang (2013)

Alice Smith

PlaceOfBirth - Seattle

PlaceOfBirth - Seattle

Science

Profession - Scientist

Profession - Scientist

PlaceOfBirth - Seattle

“list of scientists born in Seattle”

Profession - Scientist

Profession - Scientist & PlaceOfBirth - Seattle

Execute this fragment against Freebase, returns Alice Smith (and others)

Liang et al. (2011), Liang (2013)

Parsing into Lambda-DCS

Derivation $d$ on sentence $x$:

- No more explicit syntax in these derivations like we had in CCG
- Building the lexicon: more sophisticated process than GENLEX, but can handle thousands of predicates
- Log-linear model with features on rules:
  $$ P(d|x) \propto \exp \sum_{r \in d} f(r, x) $$
- Similar to CRF parsers

Berant et al. (2013)

Parsing with Lambda-DCS

- Learn just from question-answer pairs: maximize the likelihood of the right denotation $y$ with the derivation $d$ marginalized out
  $$ O(\theta) = \sum_{i=1}^{n} \log \sum_{d \in D(x):[d,z]_{K}=y_i} p_{\theta}(d \mid x_i). $$
  For each example:
  - Run beam search to get a set of derivations
  - Let $d = $ highest-scoring derivation in the beam
  - Let $d^* = $ highest-scoring derivation in the beam with correct denotation
  - Do a structured perceptron update towards $d^*$ away from $d$

Berant et al. (2013)

Learning

- Each vertical slice is the beam for one example.
  Green = correct denotation

- Only a small number of questions are even reachable by beam search initially (but some questions are very easy so even a totally untrained model can answer them)
- During training, more and more “good” derivations surface and will result in model updates

Berant et al. (2013)
Takeaways

- Can represent meaning with first order logic and lambda calculus
- Can bridge syntax and semantics and create semantic parsers that can interpret language into lambda-calculus expressions
- Useful for querying databases, question answering, etc.
- Next time: neural net methods for doing this that rely less on having explicit grammars