**Truth-Conditional Semantics**

- **Linguistic expressions:**
  - "Bob sings"
- **Logical translations:**
  - \( \text{sings(bob)} \)
  - Could be \( p_{1218}(e_{397}) \)
- **Denotation:**
  - \([\text{bob}] = \) some specific person (in some context)
  - \([\text{sings(bob)}] = ??? \)
- **Types on translations:**
  - \( \text{bob} : e \) (for entity)
  - \( \text{sings(bob)} : t \) (for truth-value)

---

**Truth-Conditional Semantics**

- **Proper names:**
  - Refer directly to some entity in the world
  - Bob : bob \( [\text{bob}] \rightarrow ??? \)
- **Sentences:**
  - Are either true or false (given how the world actually is)
  - Bob sings : \( \text{sings(bob)} \)
- **So what about verbs (and verb phrases)?**
  - \( \text{sings} \) must combine with \( \text{bob} \) to produce \( \text{sings(bob)} \)
  - The \( \lambda \)-calculus is a notation for functions whose arguments are not yet filled.
  - \( \text{sings} : \lambda x.\text{sings}(x) \)
  - This is a predicate – a function which takes an entity (type \( e \)) and produces a truth value (type \( t \)).
  - We can write its type as \( e \rightarrow t \).
  - Adjectives?
Compositional Semantics

- So now we have meanings for the words
- How do we know how to combine words?
- Associate a combination rule with each grammar rule:
  - $S : \beta(\alpha) \rightarrow NP : \alpha$, $VP : \beta$ (function application)
  - $VP : \lambda x . \alpha(x) \rightarrow \beta(\alpha) \rightarrow VP : \alpha$ and $\emptyset$, $VP : \beta$ (intersection)
- Example:

```
  S
   NP
    Bob
  VP
   and
  VP
    sings
      \lambda y.sings(y)
  \lambda z.dances(z)
  \lambda x.sings(x) \land \lambda z.dances(x)
  \lambda x.sings(x) \land \lambda z.dances(x)(bob)
  \lambda x.sings(x) \land \lambda z.dances(x)
```

Denotation

- What do we do with logical translations?
  - Translation language (logical form) has fewer ambiguities
  - Can check truth value against a database
    - Denotation ("evaluation") calculated using the database
  - More usefully: assert truth and modify a database, either explicitly or implicitly
    - eg prove a consequence from asserted axioms
  - Questions: check whether a statement in a corpus entails the (question, answer) pair:
    - "Bob sings and dances" → "Who sings?" + "Bob"
    - Chain together facts and use them for comprehension

Other Cases

- Transitive verbs:
  - likes : $\lambda x. \lambda y. \text{likes}(y,x)$
  - Two-place predicates of type $e \rightarrow (e \rightarrow t)$.
  - likes Amy : $\lambda y. \text{likes}(y,Amy)$ is just like a one-place predicate.
- Quantifiers:
  - What does "Everyone" mean here?
  - Everyone : $\lambda y. \forall x. f(x)$
  - Mostly works, but some problems
    - Have to change our NP/VP rule.
    - Won’t work for "Amy likes everyone."
    - "Everyone likes someone."
    - This gets tricky quickly!

Indefinites

- First try
  - "Bob ate a waffle" : $\text{ate}(bob,waffle)$
  - "Amy ate a waffle" : $\text{ate}(amy,waffle)$
- Can’t be right!
  - $\exists x : \text{waffle}(x) \rightarrow \text{ate}(bob,x)$
  - What does the translation of "a" have to be?
  - What about "the"?
  - What about "every"?
Grounding

- Grounding
  - Why does the translation `likes : x.λx̄likes(y,x)` have anything to do with actual liking?
  - It doesn't (unless the denotation model says so)
  - Sometimes that’s enough: wire up `bought` to the appropriate entry in a database

- Meaning postulates
  - Insist, e.g. `∀x,y.likes(y,x) → knows(y,x)`
  - This gets into lexical semantics issues

- Statistical / neural version?

Tense and Events

- In general, you don’t get far with verbs as predicates
- Better to have event variables `e`
  - “Alice danced” : `danced(alice)`
  - `∃ e : dance(e) ∧ agent(e,alice) ∧ (time(e) < now)`
- Event variables let you talk about non-trivial tense / aspect structures
  - “Alice had been dancing when Bob sneezed”
  - `∃ e, e' : dance(e) ∧ agent(e,alice) ∧ sneeze(e') ∧ agent(e',bob) ∧ (start(e) < start(e') ∧ end(e) = end(e')) ∧ (time(e') < now)`

- Minimal recursion semantics, cf “object oriented” thinking

Adverbs

- What about adverbs?
  - “Bob sings terribly”
  - `terribly(sings)(bob)`?
  - `(terribly(sings))(bob)`?
  - `∃ e present(e) ∧ type(e, singing) ∧ agent(e,bob) ∧ manner(e, terrible)`?
  - Gets complex quickly...

Propositional Attitudes

- “Bob thinks that I am a gummi bear”
  - `thinks(bob, gummi(me))`?
  - `thinks(bob, "I am a gummi bear")`?
  - `thinks(bob, "gummi(me)")`?

- Usual solution involves intensions `(^X)` which are, roughly, the set of possible worlds (or conditions) in which `X` is true

- Hard to deal with computationally
  - Modeling other agents’ models, etc
  - Can come up in even simple dialog scenarios, e.g., if you want to talk about what your bill claims you bought vs. what you actually bought
Trickier Stuff

- Non-Intersective Adjectives
  - green ball: \( x. (\text{green}(x) \land \text{ball}(x)) \)
  - fake diamond: \( x. (\text{fake}(x) \land \text{diamond}(x)) \)

- Generalized Quantifiers
  - the: \( f. \{\text{unique-member}(f)\} \)
  - all: \( \lambda f. \forall x. f(x) \rightarrow f(x) \)
  - most?
  - Could do with more general second order predicates, too (why worse?)
    - the(cat, meows), all(tat, meows)

- Generics
  - "Cats like naps"
  - "The players scored a goal"

- Pronouns (and bound anaphora)
  - "If you have a dime, put it in the meter."

- ... the list goes on and on!

Scope Ambiguities

- Quantifier scope
  - "All majors take a data science class"
  - "Someone took each of the electives"
  - "Everyone didn’t hand in their exam"

- Deciding between readings
  - Multiple ways to work this out
    - Make it syntactic (movement)
    - Make it lexical (type-shifting)

Classic Implementation, TAG, Idioms

- Add a “sem” feature to each context-free rule
  - \( S \rightarrow \text{NP loves NP} \)
  - \( S[\text{sem=loves}][x,y] \rightarrow \text{NP[sem=x]} \text{loves NP[sem=y]} \)
  - Meaning of \( S \) depends on meaning of NPs

- TAG version:
  - \( S \rightarrow \text{NP loves NP} \rightarrow \text{VP} \)
  - \( S \rightarrow \text{NP died(x) VP} \)

- Template filling: \( S[\text{sem=showflights}][x,y] \rightarrow \text{I want a flight from NP[sem=x] to NP[sem=y]} \)

Logical Form Translation
Mapping to LF: Zettlemoyer & Collins 05/07

The task:
- Input: List one way flights to Prague.
- Output: $\lambda x. \text{flight}(x) \land \text{one}_\text{way}(x) \land \text{to}(x, \text{PRG})$

Challenging learning problem:
- Derivations (or parses) are not annotated
- Approach: [Zettlemoyer & Collins 2005]
- Learn a lexicon and parameters for a weighted Combinatory Categorial Grammar (CCG)

[Slides from Luke Zettlemoyer]

Background

- Combinatory Categorial Grammar (CCG)
- Weighted CCGs
- Learning lexical entries: GENLEX

CCG Parsing

- Combinatory Categorial Grammar
  - Fully (mono-)lexicalized grammar
  - Categories encode argument sequences
  - Very closely related to the lambda calculus
  - Can have spurious ambiguities (why?)

\[
\begin{align*}
\text{John} &\rightarrow \text{NP : john} \\
\text{shares} &\rightarrow \text{NP : shares} \\
\text{boys} &\rightarrow (S\langle\text{NP}\rangle)/\text{NP} : \lambda x. \text{boys}(x) \\
\text{sleep} &\rightarrow S\langle\text{NP}\rangle : \lambda x. \text{sleep}(x) \\
\text{well} &\rightarrow (S\langle\text{NP}\rangle)/\langle S\langle\text{NP}\rangle\rangle : \lambda x. \text{well}(x)
\end{align*}
\]

CCG Lexicon

<table>
<thead>
<tr>
<th>Words</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>flights</td>
<td>$N : \lambda x. \text{flight}(x)$</td>
</tr>
<tr>
<td>to</td>
<td>$(N\langle\text{NP}\rangle)/\text{NP} : \lambda x. \text{flight}(x) \land \text{to}(y, x)$</td>
</tr>
<tr>
<td>Prague</td>
<td>NP : PRG</td>
</tr>
<tr>
<td>New York city</td>
<td>NP : NYC</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Parsing Rules (Combinators)

Application
- \( \frac{X}{Y} : f \quad \frac{Y}{a} \Rightarrow X : f(a) \)
- \( \frac{Y}{a} \quad \frac{X \backslash Y}{f} \Rightarrow X : f(a) \)

Composition
- \( \frac{X}{Y} : f \quad \frac{Y}{Z} : g \Rightarrow \frac{X}{Z} : \lambda x. f(g(x)) \)
- \( \frac{Y}{Z} : f \quad \frac{X \backslash Y}{g} \Rightarrow \frac{X}{Z} : \lambda x. f(g(x)) \)

Additional rules:
- Type Raising
- Crossed Composition

Weighted CCG

Given a log-linear model with a CCG lexicon \( \Lambda \), a feature vector \( f \), and weights \( w \):
- The best parse is:
  \[ y^* = \arg\max_y w \cdot f(x, y) \]

Where we consider all possible parses \( y \) for the sentence \( x \) given the lexicon \( \Lambda \).

CCG Parsing

<table>
<thead>
<tr>
<th>Sentence</th>
<th>Logic Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>flights</td>
<td>( \lambda x. \text{flight}(x) ) to ( (x, \text{PRG}) )</td>
</tr>
</tbody>
</table>

Lexical Generation

Input Training Example

Sentence: Show me flights to Prague.
Logic Form: \( \lambda x. \text{flight}(x) \) to \( (x, \text{PRG}) \)

Output Lexicon

<table>
<thead>
<tr>
<th>Words</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show me</td>
<td>S/N : ( \lambda f. f )</td>
</tr>
<tr>
<td>flights</td>
<td>N : ( \lambda x. \text{flight}(x) )</td>
</tr>
<tr>
<td>to</td>
<td>(N(\backslash)N)/NP : ( \lambda x. \lambda f. f(x) ) to ( (y, x) )</td>
</tr>
<tr>
<td>Prague</td>
<td>NP : ( \text{PRG} )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
GENLEX: Substrings X Categories

Input Training Example
Sentence: Show me flights to Prague.
Logic Form: \(\forall x.\text{flight}(x) \land \text{to}(x, \text{PRG})\)

Output Lexicon

All possible substrings:
- Show me flights...
- Show me flights to...

Categories created by rules that trigger on the logical form:
- \(\text{NP} : \text{PRG}\)
- \(\frac{\text{N} : \forall x.\text{flight}(x)}{\frac{(\text{S}/\text{NP})/\text{NP} : \forall x.\text{A} y.\text{to}(y, x)}}{\frac{(\text{N}/\text{N})/\text{NP} : \forall y.\forall f.\forall x.\ldots}{\ldots}}{\ldots}}\)

[Zettlemoyer & Collins 2005]

Robustness

The lexical entries that work for:

Show me the latest flight from Boston to Prague on Friday

\(\frac{\text{S}/\text{NP} \quad \text{NP}/\text{N} \quad \ldots \quad \text{N}/\text{N} \quad \text{N}/\text{N} \quad \ldots}{\ldots}\)

Will not parse:

Boston to Prague the latest on Friday

\(\frac{\text{NP} \quad \ldots \quad \text{NP}/\text{N} \quad \text{NP}/\text{N} \quad \text{N}/\text{N}}{\ldots \quad \ldots \quad \ldots \quad \ldots}\)

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

Review: Application

\(\frac{\text{X}/\text{Y} : f \quad \text{Y} : a \Rightarrow \text{X} : f(a)}{\text{Y} : a \quad \text{X}/\text{Y} : f \Rightarrow \text{X} : f(a)}\)
Disharmonic Application

- Reverse the direction of the principal category:

\[
\begin{align*}
X/Y : f & \Rightarrow X : f(a) \\
Y : a & \Rightarrow X/Y : f
\end{align*}
\]

Insert missing semantic content

- \( NF : c \Rightarrow N/N : \lambda x. f(x) \wedge p(x, c) \)

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

Training: For \( t = 1…T, i = 1…n \):

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

Step 2: Lexical Generation
- Set \( \hat{\lambda} = \Lambda \cup \text{GENLEX}(x_i, z_i) \)
- Let \( \hat{y} = \text{argmax}_{y'} w \cdot f(x_i, y') \)
- Define \( \lambda_t \) to be the lexical entries in \( y^* \)
- Set lexicon to \( \Lambda = \Lambda \cup \lambda_t \)

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

Output: Lexicon \( \Lambda \) and parameters \( w \).
Neural Encoder-Decoder Approaches

 Encoder-Decoder Models

- Can view many tasks as mapping from an input sequence of tokens to an output sequence of tokens
- Semantic parsing:
  \[ \lambda \text{state( x ) \land borders( x , e89 )} \]
- Syntactic parsing:
  \[ \text{The dog ran} \quad \rightarrow \quad \text{S (NP (DT the) (NN dog)) (VP (VBD ran))} \]
  (but what if we produce an invalid tree or one with different words?) 🙃
- Machine translation, summarization, dialogue can all be viewed in this framework as well — our examples will be MT for now

Semantic Parsing as Translation

- Prolog
- Lambda calculus
- Other DSLs

Semantic Parsing as Seq2Seq

"what states border Texas"

\[ \lambda \text{x ( state( x ) \land border( x , e89 )))} \]

- Write down a linearized form of the semantic parse, train seq2seq models to directly translate into this representation
- What are some benefits of this approach compared to grammar-based?
- What might be some concerns about this approach? How do we mitigate them?

Jia and Liang (2016)
**Problem: Lack of Inductive Bias**

"what states border Texas"  "what states border Ohio"

- Parsing-based approaches handle these the same way
- Possible divergences: features, different weights in the lexicon
- Can we get seq2seq semantic parsers to handle these the same way?
- Key idea: don’t change the model, change the data
- "Data augmentation": encode invariances by automatically generating new training examples

**Possible Solution: Data Augmentation**

Jia and Liang (2016)

Examples
("what states border texas ?", answer(“NV, (state(VG), next_to(VG, NV), const(VG, stateid(texas)))))

Rules created by ABSENTI

ROOT → (“what states border STATEID ?”,
answer(NV, (state(VG), next_to(VG, NV), const(VG, stateid(STATEID)))))
STATEID → (“texas”, “ohio”)

- Let us synthesize a "what states border ohio ?" example
- Abstract out entities: now we can “remix” examples and encode invariance to entity ID. More complicated remixes too

**Possible Solution: Copying**

<table>
<thead>
<tr>
<th></th>
<th>GEO</th>
<th>ATIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Copying</td>
<td>74.6</td>
<td>69.9</td>
</tr>
<tr>
<td>With Copying</td>
<td>85.0</td>
<td>76.3</td>
</tr>
</tbody>
</table>

- For semantic parsing, copying tokens from the input (texas) can be very useful
- Copying typically helps a bit, but attention captures most of the benefit. However, vocabulary expansion is critical for some tasks (machine translation)

Jia and Liang (2016)

**Mapping to Programs**

show me the fare from c10 to c11

```python
lambda $0 e
    | exists $1 ( and ( from $1 c10 )
    | to $1 c11 )
    | ( = ( fare $1 ) $0 )
```

[Rabinovich, Stern, Klein, 2017]
Structured Models

- Meaning representations (e.g., Python) have strong underlying syntax
- How can we **explicitly** model the underlying syntax/grammar of the target meaning representations in the decoding process?

![Abstract Syntax Trees]

**Input Intent** (x)

\[ sort\_my\_list\text{ in descending order} \]

**Generated AST** (y)

- \[ pt(y): \text{a seq2seq model with prior syntactic information} \]
- Deterministic transformation (using Python ast:oc library)

**Surface Code** (c)

\[ \text{sorted}\_my\_list, \text{reverse=True} \]

Next section includes slides from Yin / Neubig

---

AST-Structured Neural Modules

![AST-Structured Neural Modules](image)

- [Rabinovich, Stern, Klein, 2017]

AST-Structured Fragments

![AST-Structured Fragments](image)
Example Results Across Tasks

<table>
<thead>
<tr>
<th>System</th>
<th>ATIS Accuracy</th>
<th>System</th>
<th>GEO Accuracy</th>
<th>System</th>
<th>Jobs Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZH15</td>
<td>94.2</td>
<td>ZH15</td>
<td>88.9</td>
<td>ZH15</td>
<td>85.0</td>
</tr>
<tr>
<td>ZC07</td>
<td>84.6</td>
<td>KCAZ13</td>
<td>89.0</td>
<td>PEK03</td>
<td>88.0</td>
</tr>
<tr>
<td>WKZ14</td>
<td><strong>91.3</strong></td>
<td>WKZ14</td>
<td>90.4</td>
<td>LJK13</td>
<td>90.7</td>
</tr>
<tr>
<td>DL16</td>
<td>84.6</td>
<td>DL16</td>
<td>87.1</td>
<td>DL16</td>
<td>90.0</td>
</tr>
<tr>
<td>ASN</td>
<td>85.3</td>
<td>ASN</td>
<td>85.7</td>
<td>ASN</td>
<td><strong>91.4</strong></td>
</tr>
<tr>
<td>+ SUPATT</td>
<td>85.9</td>
<td>+ SUPATT</td>
<td>87.1</td>
<td>+ SUPATT</td>
<td>92.9</td>
</tr>
</tbody>
</table>

[Example Results Across Tasks](Rabinovich, Stern, Klein, 2017)

Copying / Pointer Networks

**Intent** Join app.config.path and string `'locale'` into a file path, substitute it for locatedir.

**Pred.** `locatedir = os.path.join(app.config.path, 'locale')` ✓

**Intent** Self plural is an lambda function with an argument n, which returns result of boolean expression n not equal to integer 1

**Pred.** `self.plural = lambda n: len(n) != 1` ✗

**Intent** `<name> Burly Rockjaw Trogg</name> <cost=5> <cost> <attack=3> <attack> <defense=5> <defense> <desc> Whenever your opponent casts a spell, gain 2 Attack. </desc> </desc> </attack> </attack> <rarity=Common> <rarity> ...

[Copying / Pointer Networks](Rabinovich, Stern, Klein, 2017)