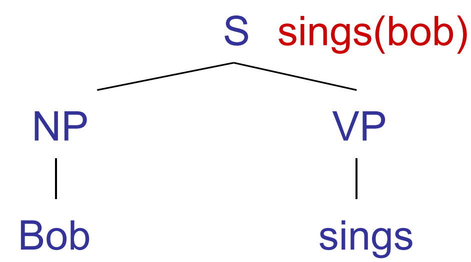
Natural Language Processing

Compositional Semantics

Dan Klein – UC Berkeley

Truth-Conditional Semantics

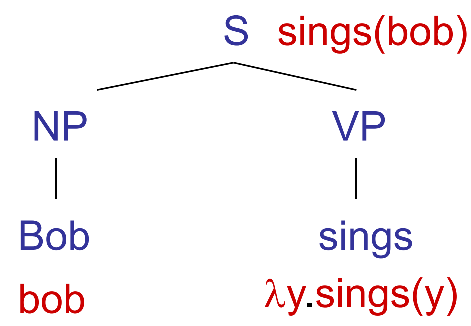
Truth-Conditional Semantics



- Linguistic expressions:
  - "Bob sings"
- Logical translations:
  - sings(bob)
  - Could be p_1218(e_397)
- Denotation:
  - [[bob]] = some specific person (in some context)
  - [[sings(bob)]] = ???
- Types on translations:
  - bob : e (for entity)
  - sings(bob) : t (for truth-value)

Truth-Conditional Semantics

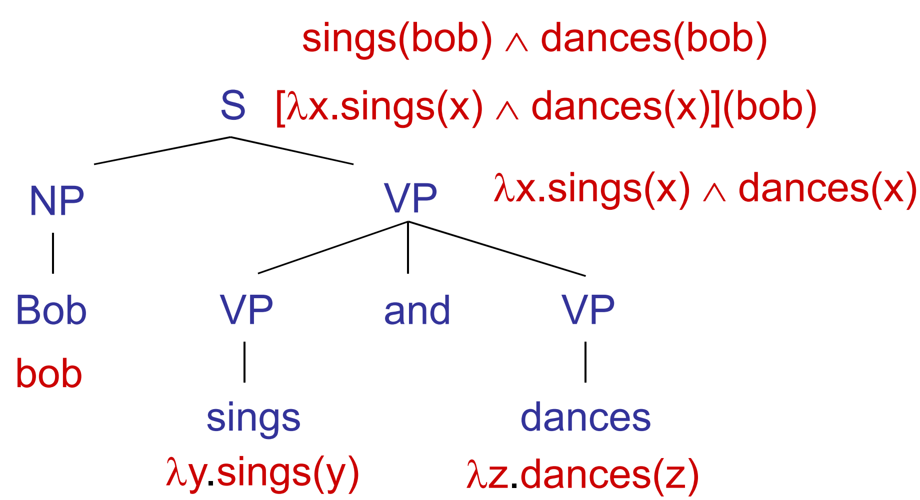
- Proper names:
  - Refer directly to some entity in the world
  - Bob : bob   [[bob]]^W → ???
- Sentences:
  - Are either true or false (given how the world actually is)
  - Bob sings : sings(bob)



- So what about verbs (and verb phrases)?
  - sings must combine with bob to produce sings(bob)
  - The λ-calculus is a notation for functions whose arguments are not yet filled.
  - sings : λx.sings(x)
  - This is *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:
  - $S : \beta(\alpha) \rightarrow NP : \alpha \quad VP : \beta$ (function application)
  - $VP : \lambda x\,.\,\alpha(x) \wedge \beta(x) \rightarrow VP : \alpha \quad and : \varnothing \quad VP : \beta$ (intersection)
- Example:

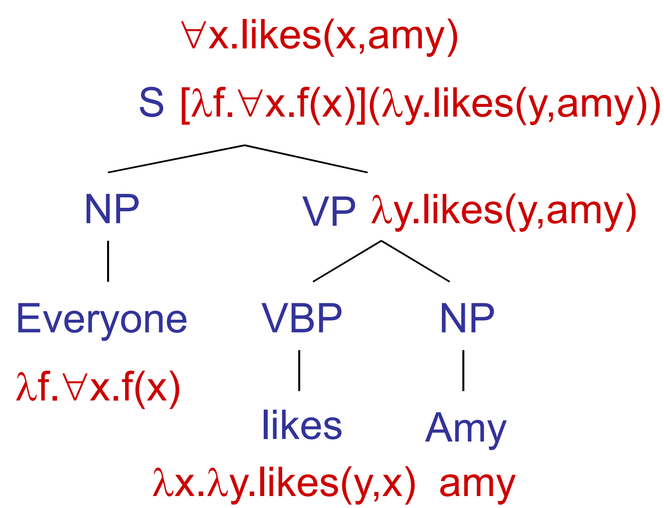


Denotation

- 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
  - 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

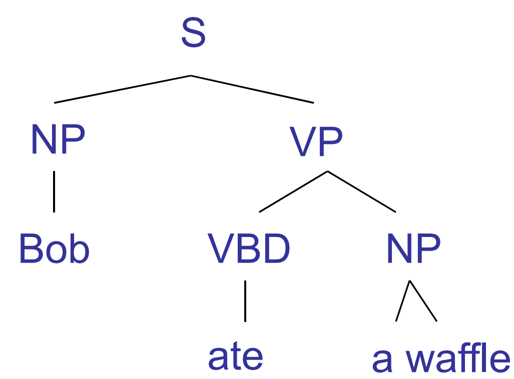
- Transitive verbs:
  - likes : λx.λy.likes(y,x)
  - Two-place predicates of type e→(e→t).
  - likes Amy : λy.likes(y,Amy) is just like a one-place predicate.
- Quantifiers:
  - What does "Everyone" mean here?
  - Everyone : λf.∀x.f(x)
  - Mostly works, but some problems
    - Have to change our NP/VP rule.
    - Won't work for "Amy likes everyone."
  - "Everyone likes someone."
  - This gets tricky quickly!



Indefinites

- First try
  - "Bob ate a waffle" : ate(bob,waffle)
  - "Amy ate a waffle" : ate(amy,waffle)

- Can't be right!
  - $\exists x : waffle(x) \wedge ate(bob,x)$
  - What does the translation of "a" have to be?
  - What about "the"?
  - What about "every"?



Grounding

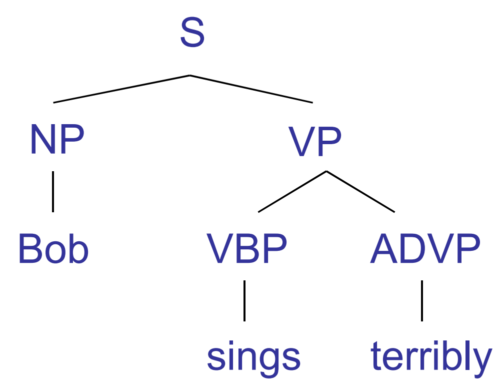
- Grounding
  - So why does the translation likes : $\lambda x.\lambda y.likes(y,x)$ have anything to do with actual liking?
  - It doesn't (unless the denotation model says so)
  - Sometimes that's enough: wire up bought to the appropriate entry in a database
- Meaning postulates
  - Insist, e.g $\forall x,y.likes(y,x) \rightarrow knows(y,x)$
  - This gets into lexical semantics issues
- Statistical 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)
  - $\exists$ e : dance(e) $\wedge$ agent(e,alice) $\wedge$ (time(e) < now)
- Event variables let you talk about non-trivial tense / aspect structures
  - "Alice had been dancing when Bob sneezed"
  - $\exists$ e, e' : dance(e) $\wedge$ agent(e,alice) $\wedge$
    sneeze(e') $\wedge$ agent(e',bob) $\wedge$
    (start(e) < start(e') $\wedge$ end(e) = end(e')) $\wedge$
    (time(e') < now)

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) ?
  - It's really not this simple…



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 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 : $\lambda x.[green(x) \wedge ball(x)]$
  - fake diamond : $\lambda x.[fake(x) \wedge diamond(x)]$? ⟶ $\lambda x.[fake(diamond(x))$
- Generalized Quantifiers
  - the : $\lambda f.[\textit{unique-member}(f)]$
  - all : $\lambda f. \lambda g\ [\forall x.f(x) \rightarrow g(x)]$
  - most?
  - Could do with more general second order predicates, too (why worse?)
    - the(cat, meows), all(cat, 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!

Multiple Quantifiers

- Quantifier scope
  - Groucho Marx celebrates quantifier order ambiguity:

    "In this country a woman gives birth every 15 min.

    Our job is to find that woman and stop her."

- Deciding between readings
  - "Bob bought a pumpkin every Halloween"
  - "Bob uses a phone as an alarm each morning"
  - Multiple ways to work this out
    - Make it syntactic (movement)
    - Make it lexical (type-shifting)

Modeling Uncertainty

- Big difference between statistical disambiguation and statistical reasoning.

The scout saw the enemy soldiers with night goggles.

  - With probabilistic parsers, can say things like "72% belief that the PP attaches to the NP."
  - That means that *probably* the enemy has night vision goggles.
  - However, you can't throw a logical assertion into a theorem prover with 72% confidence.
  - Use this to decide the expected utility of calling reinforcements?

- In short, we need probabilistic reasoning, not just probabilistic disambiguation followed by symbolic reasoning

Logical Form Translation

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?)

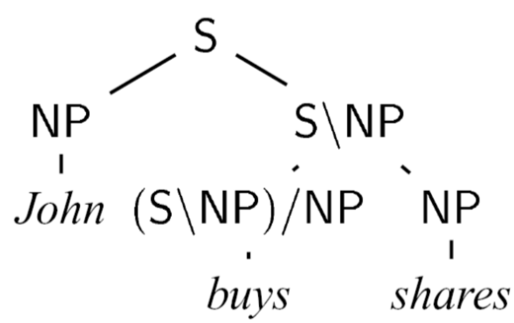
$John \vdash \mathsf{NP} : john'$

$shares \vdash \mathsf{NP} : shares'$

$buys \vdash (\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP} : \lambda x.\lambda y.buys'xy$

$sleeps \vdash \mathsf{S}\backslash\mathsf{NP} : \lambda x.sleeps'x$

$well \vdash (\mathsf{S}\backslash\mathsf{NP})\backslash(\mathsf{S}\backslash\mathsf{NP}) : \lambda f.\lambda x.well'(fx)$



Mapping to LF: Zettlemoyer & Collins 05/07

The task:

Input: `List one way flights to Prague.`

Output: $\lambda x.flight(x) \wedge one\_way(x) \wedge to(x, PRG)$

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 Lexicon

Words	Category
flights	N : $\lambda x.flight(x)$
to	(N\N)/NP : $\lambda x.\lambda f.\lambda y.f(x) \wedge to(y,x)$
Prague	NP : *PRG*
New York city	NP : *NYC*
…	…

Parsing Rules (Combinators)

Application

- `X/Y : f      Y : a   =>   X : f(a)`
- `  Y : a    X\Y : f   =>   X : f(a)`

Composition

- `X/Y : f   Y/Z : g   =>  X/Z : λx.f(g(x))`
- `Y\Z : f   X\Y : g   =>  X\Z : λx.f(g(x))`

Additional rules:

- Type Raising
- Crossed Composition

CCG Parsing

Show me	flights	to	Prague
S/N	N	(N\N)/NP	NP
$\lambda f.f$	$\lambda x.flight(x)$	$\lambda y.\lambda f.\lambda x.f(y) \wedge to(x,y)$	PRG

N\N

$\lambda f.\lambda x.f(x) \wedge to(x,PRG)$

N

$\lambda x.flight(x) \wedge to(x,PRG)$

S

$\lambda x.flight(x) \wedge to(x,PRG)$

Weighted CCG

Given a log-linear model with a CCG lexicon Λ, a feature vector *f*, and weights *w*.

- The best parse is:

$$y^* = \operatorname*{argmax}_y \; w \cdot f(x,y)$$

Where we consider all possible parses *y* for the sentence *x* given the lexicon Λ.

Lexical Generation

Input Training Example

Sentence: `Show me flights to Prague.`

Logic Form: $\lambda x.flight(x) \wedge to(x, PRG)$

Output Lexicon

Words	Category
Show me	$S/N : \lambda f.f$
flights	$N : \lambda x.flight(x)$
to	$(N\backslash N)/NP : \lambda x.\lambda f.\lambda y.f(x) \wedge to(y, x)$
Prague	$NP : PRG$
...	...

GENLEX: Substrings X Categories

Input Training Example

Sentence:	`Show me flights to Prague.`
Logic Form:	$\lambda x.flight(x) \wedge to(x,PRG)$

Output Lexicon

All possible substrings:

```
Show
me
flights
…
Show me
Show me flights
Show me flights to
…
```

X

Categories created by rules that trigger on the logical form:

NP : PRG

N : $\lambda x.flight(x)$

(S\NP)/NP : $\lambda x.\lambda y.to(y,x)$

(N\N)/NP : $\lambda y.\lambda f.\lambda x.$ …

…

[Zettlemoyer & Collins 2005]

Robustness

The lexical entries that work for:

```
Show me   the latest   flight   from Boston   to Prague   on Friday
  S/NP       NP/N         N          N\N          N\N          N\N
   …          …           …           …            …            …
```

Will not parse:

```
Boston to Prague the latest on Friday
  NP      N\N        NP/N        N\N
  …       …          …           …
```

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

```
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 — N, *λx.flight(x)*

one way — N/N, *λf.λx.f(x)∧one_way(x)*

N

λx.flight(x)∧one_way(x)

Missing content words

Insert missing semantic content

- NP : c => N\N : $\lambda f.\lambda x.f(x) \wedge p(x,c)$

flights	Boston	to Prague
N $\lambda x.flight(x)$	NP BOS	N\N $\lambda f.\lambda x.f(x) \wedge to(x, PRG)$
	N\N $\lambda f.\lambda x.f(x) \wedge from(x, BOS)$	

N
$\lambda x.flight(x) \wedge from(x, BOS)$

N
$\lambda x.flight(x) \wedge from(x, BOS) \wedge to(x, PRG)$

Missing content-free words

Bypass missing nouns

- N\N : f => N : f(λx.true)

Northwest Air	to Prague
N/N	N\N
$\lambda f.\lambda x.f(x) \wedge airline(x, NWA)$	$\lambda f.\lambda x.f(x) \wedge to(x, PRG)$
	N
	$\lambda x.to(x, PRG)$

N

$$\lambda x.airline(x, NWA) \wedge to(x, PRG)$$

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

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

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

Step 2: Lexical Generation

- Set $\lambda = \Lambda \cup \text{GENLEX}(x_i, z_i)$
- Let $\hat{y} = \arg \max_{y \text{ s.t. } L(y)=z_i} w \cdot f(x_i, y)$
- Define λ_i to be the lexical entries in y^
- Set lexicon to $\Lambda = \Lambda \cup \lambda_i$

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 Λ and parameters w.

Related Work for Evaluation

Hidden Vector State Model: He and Young 2006

- Learns a probabilistic push-down automaton with EM
- Is integrated with speech recognition

λ-WASP: Wong & Mooney 2007

- Builds a synchronous CFG with statistical machine translation techniques
- Easily applied to different languages

Zettlemoyer and Collins 2005

- Uses GENLEX with maximum likelihood batch training and stricter 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

$$\lambda x.flight(x) \wedge from(x,BOS) \wedge to(x,PRG)$$

is represented as:

{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

ATIS Test Set [Z+C 2007]

Exact Match Accuracy:

	Precision	Recall	F1
Single-Pass	90.61	81.92	**86.05**
Two-Pass	85.75	84.60	85.16

Geo880 Test Set

Exact Match Accuracy:

	Precision	Recall	F1
Single-Pass	95.49	83.20	**88.93**
Two-Pass	91.63	86.07	88.76
Zettlemoyer & Collins 2005	96.25	79.29	86.95
Wong & Mooney 2007	93.72	80.00	86.31