Natural Language Processing

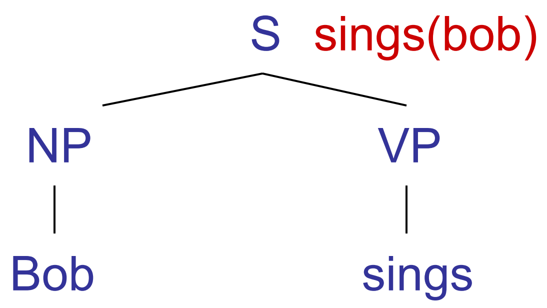
Berkeley NLP

Compositional Semantics

Truth-Conditional Semantics

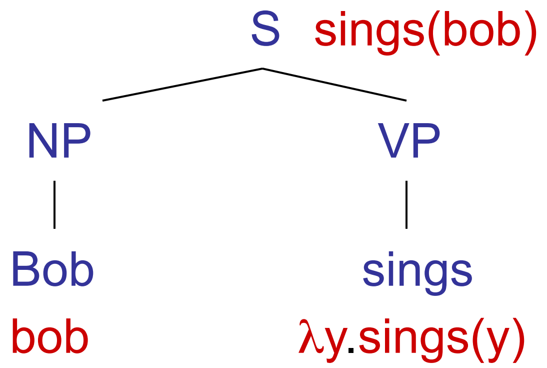
Truth-Conditional Semantics

- 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

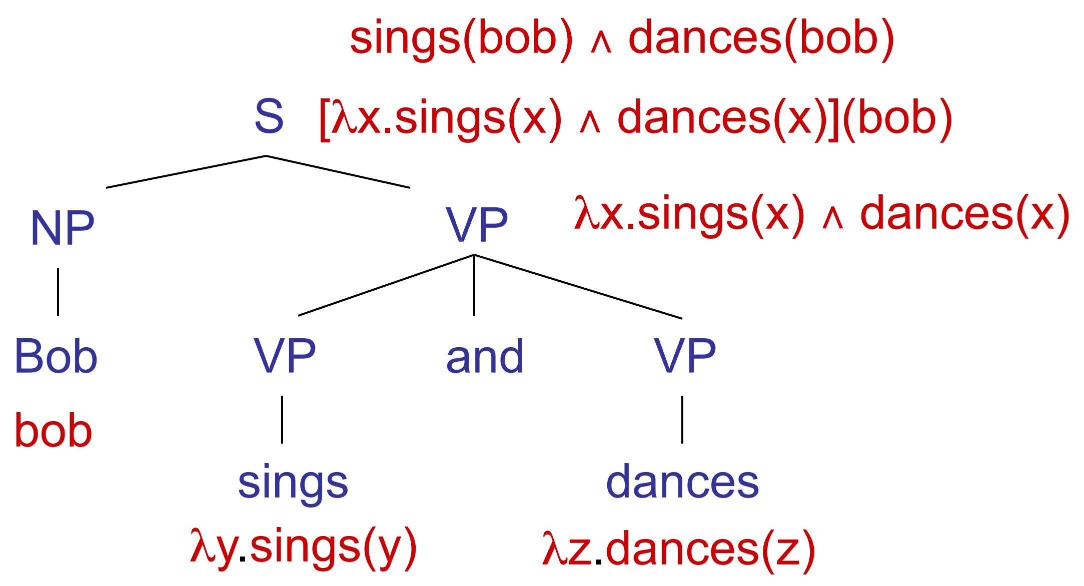
- Proper names:
  - Refer directly to some entity in the world
  - Bob : bob $[[bob]]^W \rightarrow$???
- Sentences:
  - Are either true or false (given how the world actually is)
  - Bob sings : sings(bob)



- 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 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 : α VP : β (function application)
  - VP : $\lambda x . \alpha(x) \wedge \beta(x) \rightarrow$ VP : α and : $\varnothing$ VP : β (intersection)
- Example:

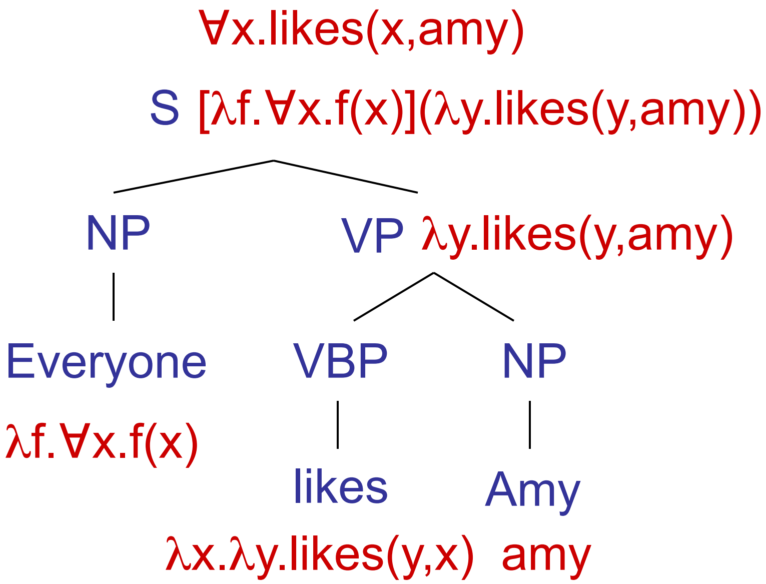


Denotation

- 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
  - Or the opposite: 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

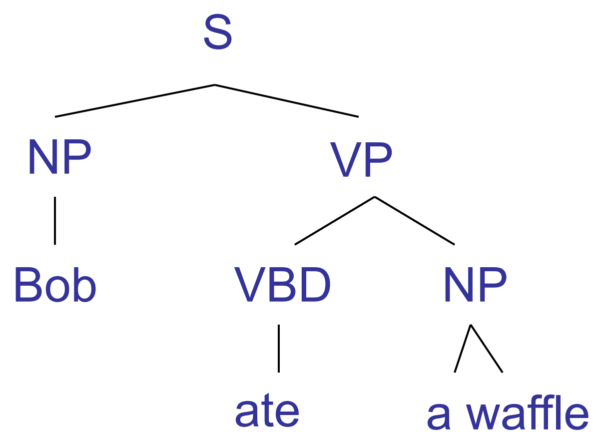
- Transitive verbs:
  - likes : $\lambda x.\lambda y.likes(y,x)$
  - Two-place predicates of type $e \rightarrow (e \rightarrow t)$.
  - likes Amy : $\lambda y.likes(y,Amy)$ is just like a one-place predicate.
- Quantifiers:
  - What does “Everyone” mean here?
  - Everyone : $\lambda f.\forall x.f(x)$
  - 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” : ate(bob,waffle)
  - “Amy ate a waffle” : ate(amy,waffle)

- Can’t be right!
  - ∃ x : waffle(x) ∧ 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 / 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)
  - $\exists\, e : dance(e) \wedge agent(e,alice) \wedge (time(e) < now)$
- Event variables let you talk about non-trivial tense / aspect structures
  - "Alice had been dancing when Bob sneezed"
  - $\exists\, e, e' : \quad dance(e) \wedge agent(e,alice) \wedge$

    $sneeze(e') \wedge agent(e',bob) \wedge$

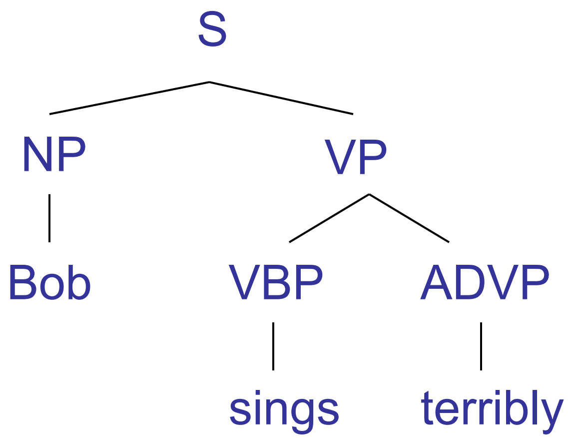
    $(start(e) < start(e') \wedge end(e) = end(e')) \wedge$

    $(time(e') < now)$

- 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 : λx.[green(x) ∧ ball(x)]
  - fake diamond : λx.[fake(x) ∧ diamond(x)] ? → λx.[fake(diamond(x))
- Generalized Quantifiers
  - the : λf.[*unique-member*(f)]
  - all : λf. λg [∀x.f(x) → 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!

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)

Logical Form Translation

Mapping to LF: Zettlemoyer & Collins 05/07

The task:

Input: `List one way flights to Prague.`

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

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 Parsing

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

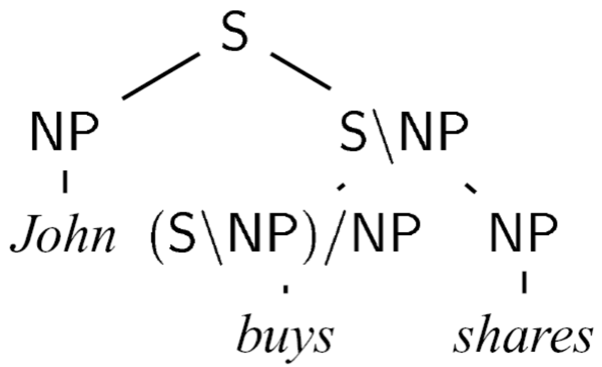
$$John \vdash \mathsf{NP} : john'$$

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

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

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

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



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

to Prague:

N\N

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

flights to Prague:

N

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

Show me flights to Prague:

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:

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\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]

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
- If $L(y^*) = z_i$, go to the next example

Step 2: Lexical Generation

- Set
- Let
- Define λ_i to be the lexical entries in $y^\wedge$
- Set lexicon to $\Lambda = \Lambda \cup \lambda_i$

Step 3: Update Parameters

- Let
- If
    - Set

Output: Lexicon Λ and parameters w.