Natural Language Processing (CSE 490U): Compositional Semantics

Noah Smith

© 2017

University of Washington
nasmith@cs.washington.edu

March 1, 2017

Bridging the Gap between Language and the World

In order to link NL to a knowledge base, we might want to design a formal way to represent meaning.
Desiderata for a **meaning representation language**:

Bridging the Gap between Language and the World

In order to link NL to a knowledge base, we might want to design a formal way to represent meaning.

Desiderata for a **meaning representation language**:

- represent the state of the world, i.e., a knowledge base

Bridging the Gap between Language and the World

In order to link NL to a knowledge base, we might want to design a formal way to represent meaning.

Desiderata for a **meaning representation language**:

- represent the state of the world, i.e., a knowledge base
- query the knowledge base (e.g., verify that a statement is true, or answer a question)

Bridging the Gap between Language and the World

In order to link NL to a knowledge base, we might want to design a formal way to represent meaning.

Desiderata for a **meaning representation language**:

- represent the state of the world, i.e., a knowledge base
- query the knowledge base (e.g., verify that a statement is true, or answer a question)
- handle ambiguity, vagueness, and non-canonical forms
  - "I wanna eat someplace that's close to UW"
  - "something not too spicy"

Bridging the Gap between Language and the World

In order to link NL to a knowledge base, we might want to design a formal way to represent meaning.

Desiderata for a **meaning representation language**:

- represent the state of the world, i.e., a knowledge base
- query the knowledge base (e.g., verify that a statement is true, or answer a question)
- handle ambiguity, vagueness, and non-canonical forms
  - "I wanna eat someplace that's close to UW"
  - "something not too spicy"
- support inference and reasoning
  - "can Karen eat at Schultzy's?"

Bridging the Gap between Language and the World

In order to link NL to a knowledge base, we might want to design a formal way to represent meaning.
Desiderata for a **meaning representation language**:

- represent the state of the world, i.e., a knowledge base
- query the knowledge base (e.g., verify that a statement is true, or answer a question)
- handle ambiguity, vagueness, and non-canonical forms
  - "I wanna eat someplace that's close to UW"
  - "something not too spicy"
- support inference and reasoning
  - "can Karen eat at Schultzy's?"

Eventually (but not today):

- deal with non-literal meanings
- expressiveness across a wide range of subject matter

A (Tiny) World Model

- **Domain:** Adrian, Brook, Chris, Donald, Schultzy's Sausage, Din Tai Fung, Banana Leaf, American, Chinese, Thai
- **Property:** Din Tai Fung has a long wait, Schultzy's is noisy; Alice, Bob, and Charles are human
- **Relations:** Schultzy's serves American, Din Tai Fung serves Chinese, and Banana Leaf serves Thai

Simple questions are easy:

- Is Schultzy's noisy?
- Does Din Tai Fung serve Thai?

A (Tiny) World Model

- **Domain:** Adrian, Brook, Chris, Donald, Schultzy's Sausage, Din Tai Fung, Banana Leaf, American, Chinese, Thai
  $a, b, c, d, ss, dtf, bl, am, ch, th$
- **Property:** Din Tai Fung has a long wait, Schultzy's is noisy; Alice, Bob, and Charles are human
  $Longwait = \{dtf\}, Noisy = \{ss\}, Human = \{a, b, c\}$
- **Relations:** Schultzy's serves American, Din Tai Fung serves Chinese, and Banana Leaf serves Thai
  $Serves = \{(ss, am), (dtf, ch), (bl, th)\}, Likes = \{(a, ss), (a, dtf), \ldots\}$

Simple questions are easy:

- Is Schultzy's noisy?
- Does Din Tai Fung serve Thai?

A Quick Tour of First-Order Logic

- **Term:** a constant (ss) or a variable
- **Formula:** defined inductively ...
  - If R is an n-ary relation and $t_1, \ldots, t_n$ are terms, then $R(t_1, \ldots, t_n)$ is a formula.
  - If ϕ is a formula, then its negation, $\neg\phi$, is a formula.
  - If ϕ and ψ are formulas, then binary logical connectives can be used to create formulas:
    - $\phi \wedge \psi$
    - $\phi \vee \psi$
    - $\phi \Rightarrow \psi$
    - $\phi \oplus \psi$
  - If ϕ is a formula and v is a variable, then quantifiers can be used to create formulas:
    - Universal quantifier: $\forall v, \phi$
    - Existential quantifier: $\exists v, \phi$

Note: Leaving out functions, because we don't need them in a single lecture on FOL for NL.

Translating Between FOL and NL

1. Schultzy's is not loud
2. Some human likes Chinese
3. If a person likes Thai, then they aren't friends with Donald
4. $\forall x, Restaurant(x) \Rightarrow (Longwait(x) \vee \neg Likes(a, x))$
5. $\forall x, \exists y, \neg Likes(x, y)$
6. $\exists y, \forall x, \neg Likes(x, y)$

Translating Between FOL and NL

1. Schultzy's is not loud $\neg Noisy(ss)$
2. Some human likes Chinese
3. If a person likes Thai, then they aren't friends with Donald
4. $\forall x, Restaurant(x) \Rightarrow (Longwait(x) \vee \neg Likes(a, x))$
5. $\forall x, \exists y, \neg Likes(x, y)$
6. $\exists y, \forall x, \neg Likes(x, y)$

Translating Between FOL and NL

1. Schultzy's is not loud $\neg Noisy(ss)$
2. Some human likes Chinese $\exists x, Human(x) \wedge Likes(x, ch)$
3. If a person likes Thai, then they aren't friends with Donald
4. $\forall x, Restaurant(x) \Rightarrow (Longwait(x) \vee \neg Likes(a, x))$
5. $\forall x, \exists y, \neg Likes(x, y)$
6. $\exists y, \forall x, \neg Likes(x, y)$

Translating Between FOL and NL

1. Schultzy's is not loud $\neg Noisy(ss)$
2. Some human likes Chinese $\exists x, Human(x) \land Likes(x, ch)$
3. If a person likes Thai, then they aren't friends with Donald
   $\forall x, Human(x) \land Likes(x, th) \Rightarrow \neg Friends(x, d)$
4. $\forall x, Restaurant(x) \Rightarrow (Longwait(x) \lor \neg Likes(a, x))$
5. $\forall x, \exists y, \neg Likes(x, y)$
6. $\exists y, \forall x, \neg Likes(x, y)$

Translating Between FOL and NL

1. Schultzy's is not loud $\neg Noisy(ss)$
2. Some human likes Chinese $\exists x, Human(x) \wedge Likes(x, ch)$
3. If a person likes Thai, then they aren't friends with Donald
   $\forall x, Human(x) \wedge Likes(x, th) \Rightarrow \neg Friends(x, d)$
4. $\forall x, Restaurant(x) \Rightarrow (Longwait(x) \vee \neg Likes(a, x))$
   Every restaurant has a long wait or is disliked by Adrian.
5. $\forall x, \exists y, \neg Likes(x, y)$
6. $\exists y, \forall x, \neg Likes(x, y)$

Translating Between FOL and NL

1. Schultzy's is not loud $\neg Noisy(ss)$
2. Some human likes Chinese $\exists x, Human(x) \wedge Likes(x, ch)$
3. If a person likes Thai, then they aren't friends with Donald
   $\forall x, Human(x) \wedge Likes(x, th) \Rightarrow \neg Friends(x, d)$
4. $\forall x, Restaurant(x) \Rightarrow (Longwait(x) \vee \neg Likes(a, x))$
   Every restaurant has a long wait or is disliked by Adrian.
5. $\forall x, \exists y, \neg Likes(x, y)$
   Everybody has something they don't like.
6. $\exists y, \forall x, \neg Likes(x, y)$

Translating Between FOL and NL

1. Schultzy's is not loud $\neg Noisy(ss)$
2. Some human likes Chinese $\exists x, Human(x) \wedge Likes(x, ch)$
3. If a person likes Thai, then they aren't friends with Donald
   $\forall x, Human(x) \wedge Likes(x, th) \Rightarrow \neg Friends(x, d)$
4. $\forall x, Restaurant(x) \Rightarrow (Longwait(x) \vee \neg Likes(a, x))$
   Every restaurant has a long wait or is disliked by Adrian.
5. $\forall x, \exists y, \neg Likes(x, y)$
   Everybody has something they don't like.
6. $\exists y, \forall x, \neg Likes(x, y)$
   There exists something that nobody likes.

Logical Semantics

(Montague, 1970)

The denotation of a NL sentence is the set of conditions that must hold in the (model) world for the sentence to be true.

Every restaurant has a long wait or Adrian doesn't like it.

is true if and only if

$$\forall x, Restaurant(x) \Rightarrow (Longwait(x) \vee \neg Likes(a, x))$$

is true.

This is sometimes called the **logical form** of the NL sentence.

The Principle of Compositionality

The meaning of a NL phrase is determined by the meanings of its sub-phrases.

The Principle of Compositionality

The meaning of a NL phrase is determined by the meanings of its sub-phrases.

I.e., semantics is derived from syntax.

The Principle of Compositionality

The meaning of a NL phrase is determined by the meanings of its sub-phrases.

I.e., semantics is derived from syntax.

We need a way to express semantics of phrases, and compose them together!

λ-Calculus

(Much more powerful than what we'll see today; ask your PL professor!)

Informally, two extensions:

- λ-**abstraction** is another way to "scope" variables.
  - If ϕ is a FOL formula and v is a variable, then $\lambda v.\phi$ is a λ-term, meaning: an unnamed function from values (of v) to formulas (usually involving v)
- **application** of such functions: if we have $\lambda v.\phi$ and ψ, then $[\lambda v.\phi](\psi)$ is a formula.
  - It can be **reduced** by substituting ψ in for every instance of v in ϕ.

λ-Calculus

(Much more powerful than what we'll see today; ask your PL professor!)

Informally, two extensions:

- λ-**abstraction** is another way to "scope" variables.
  - If ϕ is a FOL formula and v is a variable, then $\lambda v.\phi$ is a λ-term, meaning: an unnamed function from values (of v) to formulas (usually involving v)
- **application** of such functions: if we have $\lambda v.\phi$ and ψ, then $[\lambda v.\phi](\psi)$ is a formula.
  - It can be **reduced** by substituting ψ in for every instance of v in ϕ.

Example:
$\lambda x.Likes(x, dtf)$ maps things to statements that they like Din Tai Fung

λ-Calculus

(Much more powerful than what we'll see today; ask your PL professor!)

Informally, two extensions:

- λ-**abstraction** is another way to "scope" variables.
  - If ϕ is a FOL formula and v is a variable, then $\lambda v.\phi$ is a λ-term, meaning: an unnamed function from values (of v) to formulas (usually involving v)
- **application** of such functions: if we have $\lambda v.\phi$ and ψ, then $[\lambda v.\phi](\psi)$ is a formula.
  - It can be **reduced** by substituting ψ in for every instance of v in ϕ.

Example:
$[\lambda x.Likes(x, dtf)](c)$ reduces to $Likes(c, dtf)$

λ-Calculus

(Much more powerful than what we'll see today; ask your PL professor!)

Informally, two extensions:

- λ-**abstraction** is another way to "scope" variables.
  - If ϕ is a FOL formula and v is a variable, then $\lambda v.\phi$ is a λ-term, meaning: an unnamed function from values (of v) to formulas (usually involving v)
- **application** of such functions: if we have $\lambda v.\phi$ and ψ, then $[\lambda v.\phi](\psi)$ is a formula.
  - It can be **reduced** by substituting ψ in for every instance of v in ϕ.

Example:
$\lambda x.\lambda y.Friends(x, y)$ maps things x to maps of things y to statements that x and y are friends

λ-Calculus

(Much more powerful than what we'll see today; ask your PL professor!)

Informally, two extensions:

- λ-**abstraction** is another way to "scope" variables.
  - If ϕ is a FOL formula and v is a variable, then $\lambda v.\phi$ is a λ-term, meaning: an unnamed function from values (of v) to formulas (usually involving v)
- **application** of such functions: if we have $\lambda v.\phi$ and ψ, then $[\lambda v.\phi](\psi)$ is a formula.
  - It can be **reduced** by substituting ψ in for every instance of v in ϕ.

Example:
$[\lambda x.\lambda y.Friends(x, y)](b)$ reduces to $\lambda y.Friends(b, y)$

λ-Calculus

(Much more powerful than what we'll see today; ask your PL professor!)

Informally, two extensions:

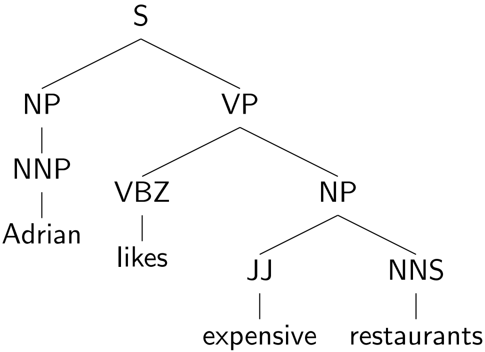
- λ-**abstraction** is another way to "scope" variables.
  - If ϕ is a FOL formula and v is a variable, then $\lambda v.\phi$ is a λ-term, meaning: an unnamed function from values (of v) to formulas (usually involving v)
- **application** of such functions: if we have $\lambda v.\phi$ and ψ, then $[\lambda v.\phi](\psi)$ is a formula.
  - It can be **reduced** by substituting ψ in for every instance of v in ϕ.

Example:
$[[\lambda x.\lambda y.Friends(x, y)](b)](a)$ reduces to $[\lambda y.Friends(b, y)](a)$, which reduces to $Friends(b, a)$

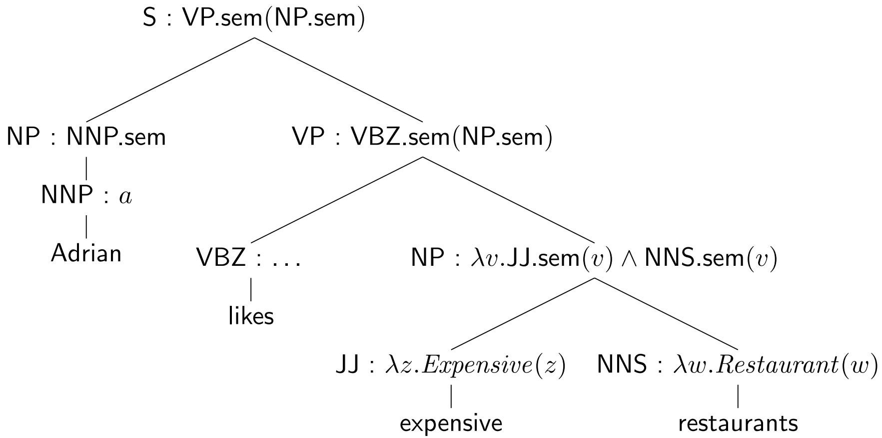
Semantic Attachments to CFG

- NNP $\rightarrow$ Adrian $\{a\}$
- VBZ $\rightarrow$ likes $\{\lambda f.\lambda y.\forall x f(x) \Rightarrow Likes(y,x)\}$
- JJ $\rightarrow$ expensive $\{\lambda x.Expensive(x)\}$
- NNS $\rightarrow$ restaurants $\{\lambda x.Restaurant(x)\}$
- NP $\rightarrow$ NNP {NNP.sem}
- NP $\rightarrow$ JJ NNS $\{\lambda x.\text{JJ.sem}(x) \wedge \text{NNS.sem}(x)\}$
- VP $\rightarrow$ VBZ NP {VBZ.sem(NP.sem)}
- S $\rightarrow$ NP VP {VP.sem(NP.sem)}

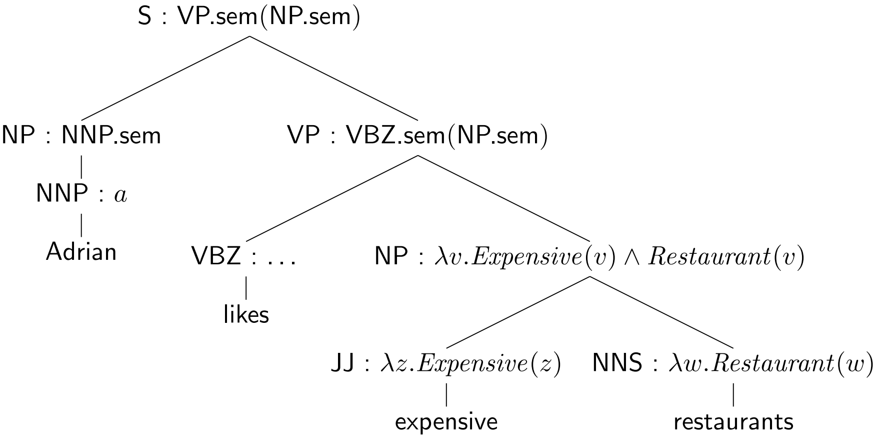
Example



Example

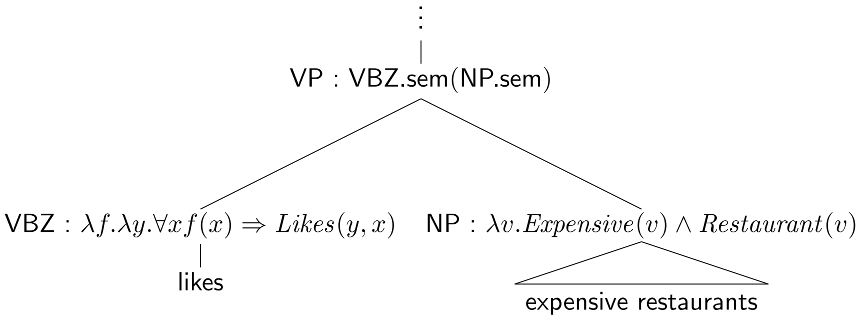

S : VP.sem(NP.sem)
NP : NNP.sem
VP : VBZ.sem(NP.sem)
NNP : a
Adrian
VBZ : . . .
likes
NP : λv.JJ.sem(v) $\wedge$ NNS.sem(v)
JJ : $\lambda z.Expensive(z)$
expensive
NNS : $\lambda w.Restaurant(w)$
restaurants

Example



$$\lambda v. \left[\underbrace{\lambda z.Expensive(z)}_{\text{JJ.sem}} \right] (v) \wedge \left[\underbrace{\lambda w.Restaurant(w)}_{\text{NNS.sem}} \right] (v)$$

Example


⋮
VP : VBZ.sem(NP.sem)
VBZ : $\lambda f.\lambda y.\forall x\, f(x) \Rightarrow Likes(y, x)$
NP : $\lambda v.Expensive(v) \wedge Restaurant(v)$
likes
expensive restaurants

Example

$\vdots$

VP : $\lambda y.\forall x, \mathit{Expensive}(x) \wedge \mathit{Restaurant}(x) \Rightarrow \mathit{Likes}(y,x)$

VBZ : $\lambda f.\lambda y.\forall x\, f(x) \Rightarrow \mathit{Likes}(y,x)$

likes

NP : $\lambda v.\mathit{Expensive}(v) \wedge \mathit{Restaurant}(v)$

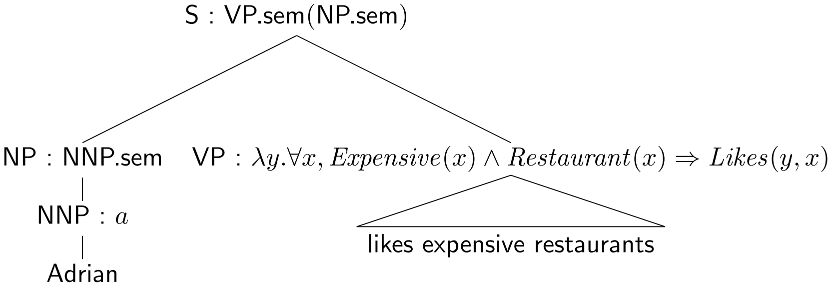
expensive restaurants

$$\left[\underbrace{\lambda f.\lambda y.\forall x\, f(x) \Rightarrow \mathit{Likes}(y,x)}_{\text{VBZ.sem}}\right]\left(\underbrace{\lambda v.\mathit{Expensive}(v) \wedge \mathit{Restaurant}(v)}_{\text{NP.sem}}\right)$$

$$\lambda y.\forall x\, [\lambda v.\mathit{Expensive}(v) \wedge \mathit{Restaurant}(v)]\,(x) \Rightarrow \mathit{Likes}(y,x)$$

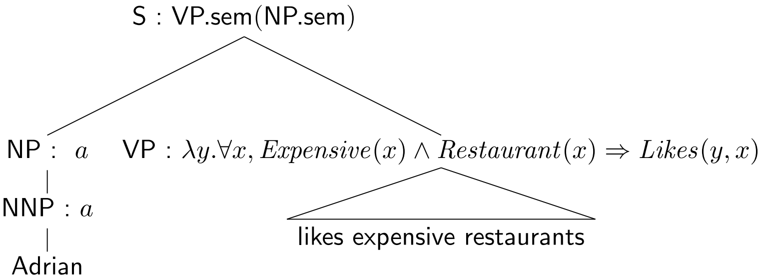
$$\lambda y.\forall x, \mathit{Expensive}(x) \wedge \mathit{Restaurant}(x) \Rightarrow \mathit{Likes}(y,x)$$

Example



S : VP.sem(NP.sem)
NP : NNP.sem
NNP : a
Adrian
VP : $\lambda y.\forall x, Expensive(x) \wedge Restaurant(x) \Rightarrow Likes(y, x)$
likes expensive restaurants

Example


S : VP.sem(NP.sem)
NP : a
VP : $\lambda y.\forall x, Expensive(x) \wedge Restaurant(x) \Rightarrow Likes(y, x)$
NNP : a
Adrian
likes expensive restaurants

Example

S : $\forall x, \mathit{Expensive}(x) \wedge \mathit{Restaurant}(x) \Rightarrow \mathit{Likes}(a, x)$

NP : a VP : $\lambda y.\forall x, \mathit{Expensive}(x) \wedge \mathit{Restaurant}(x) \Rightarrow \mathit{Likes}(y, x)$

NNP : a

Adrian

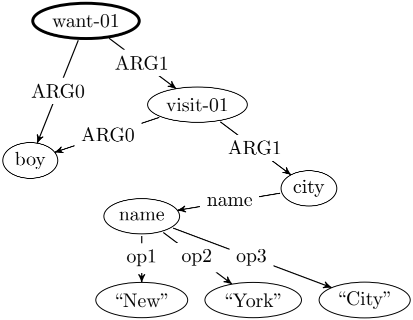
likes expensive restaurants

$$\left[\underbrace{\lambda y.\forall x, \mathit{Expensive}(x) \wedge \mathit{Restaurant}(x) \Rightarrow \mathit{Likes}(y, x)}_{\text{VP.sem}}\right]\left(\underbrace{a}_{\text{NP.sem}}\right)$$

$$\forall x, \mathit{Expensive}(x) \wedge \mathit{Restaurant}(x) \Rightarrow \mathit{Likes}(a, x)$$

Graph-Based Representations

Abstract Meaning Representation (Banarescu et al., 2013)



"The boy wants to visit New York City."

Designed for (1) annotation-ability and (2) eventual use in machine translation.

Combinatory Categorial Grammar

(Steedman, 2000)

CCG is a grammatical formalism that is well-suited for tying together syntax and semantics.

Formally, it is more powerful than CFG—it can represent some of the context-*sensitive* languages (which we do not have time to define formally).

CCG Types

Instead of the "$\mathcal{N}$" of CFGs, CCGs can have an infinitely large set of structured categories (called **types**).

- Primitive types: typically S, NP, N, and maybe more
- Complex types, built with "slashes," for example:
  - S/NP is "an S, except that it lacks an NP to the right"
  - S\NP is "an S, except that it lacks an NP to its left"
  - (S\NP)/NP is "an S, except that it lacks an NP to its right, and its left"

You can think of complex types as functions, e.g., S/NP maps NPs to Ss.

CCG Combinators

Instead of the production rules of CFGs, CCGs have a very small set of generic **combinators** that tell us how we can put types together.

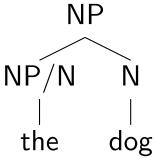
Convention writes the rule differently from CFG: $X \quad Y \Rightarrow Z$ means that X and Y combine to form a Z (the "parent" in the tree).

Application Combinator

Forward ($X/Y \quad Y \Rightarrow X$) and backward ($Y \quad X\backslash Y \Rightarrow X$)

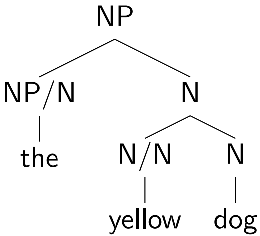
Application Combinator

Forward ($X/Y \quad Y \Rightarrow X$) and backward ($Y \quad X\backslash Y \Rightarrow X$)



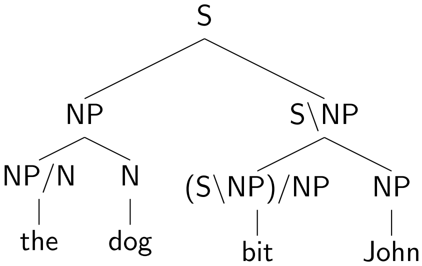
Application Combinator

Forward ($X/Y \quad Y \Rightarrow X$) and backward ($Y \quad X\backslash Y \Rightarrow X$)



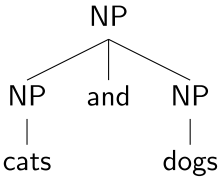
Application Combinator

Forward ($X/Y \quad Y \Rightarrow X$) and backward ($Y \quad X\backslash Y \Rightarrow X$)



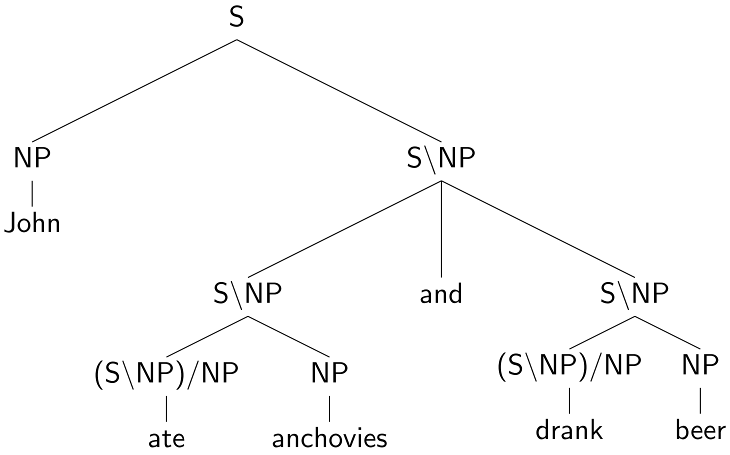
Conjunction Combinator

X and $X \Rightarrow X$



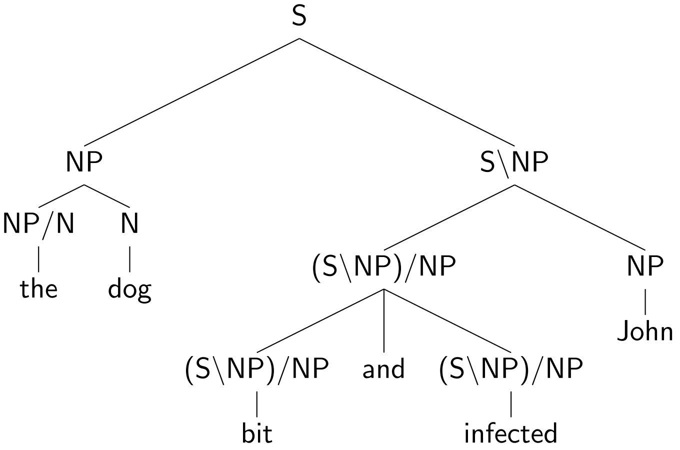
Conjunction Combinator

X and $X \Rightarrow X$



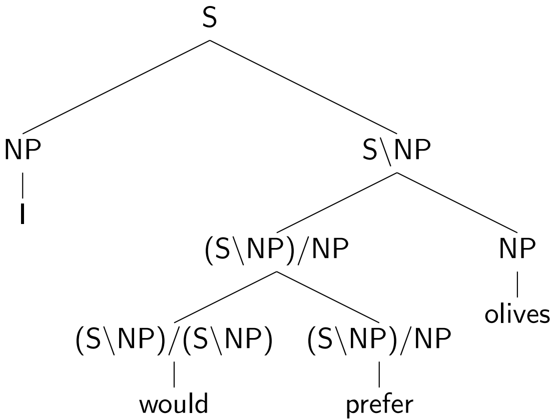
Conjunction Combinator

X and $X \Rightarrow X$



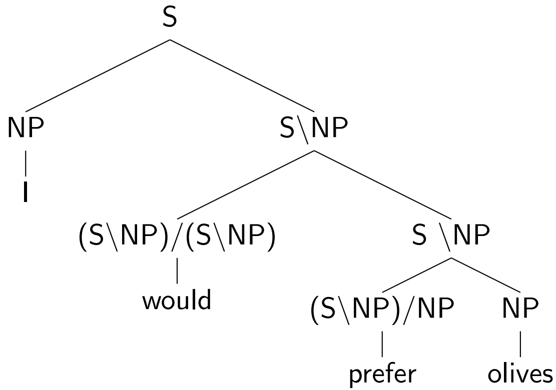
Composition Combinator

Forward ($X/Y \quad Y/Z \Rightarrow X/Z$) and backward ($Y\backslash Z \quad X\backslash Y \Rightarrow X\backslash Z$)



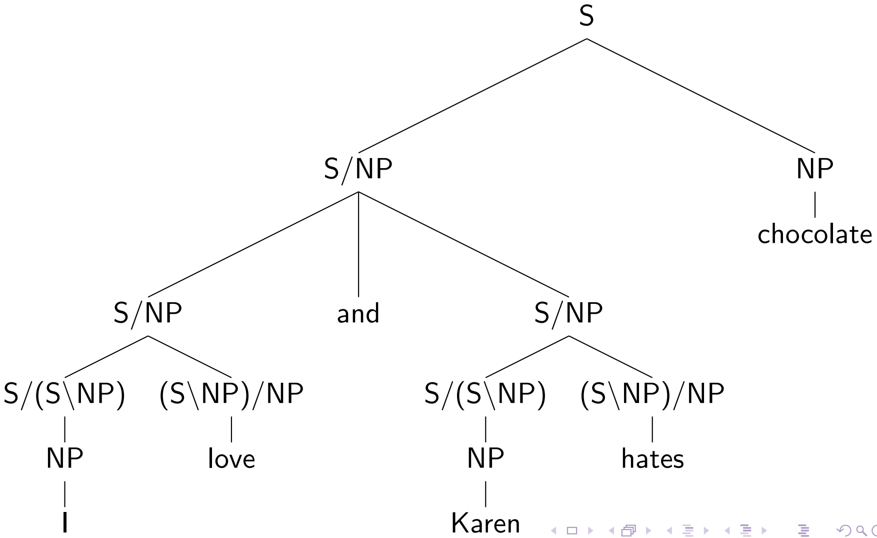
Composition Combinator

Forward $(X/Y \quad Y/Z \Rightarrow X/Z)$ and backward $(Y\backslash Z \quad X\backslash Y \Rightarrow X\backslash Z)$



Type-Raising Combinator

Forward ($X \Rightarrow Y/(Y\backslash X)$) and backward ($X \Rightarrow Y\backslash(Y/X)$)



Back to Semantics

Each combinator also tells us what to do with the semantic attachments.

- Forward application: $X/Y : f \quad Y : g \Rightarrow X : f(g)$
- Forward composition:
  $X/Y : f \quad Y/Z : g \Rightarrow X/Z : \lambda x.f(g(x))$
- Forward type-raising: $X : g \Rightarrow Y/(Y\backslash X) : \lambda f.f(g)$

CCG Lexicon

Most of the work is done in the lexicon!

Syntactic and semantic information is much more formal here.

- Slash categories define where all the syntactic arguments are expected to be
- λ-expressions define how the expected arguments get "used" to build up a FOL expression

Extensive discussion: Carpenter (1997)

Some Topics We Don't Have Time For

- Tasks, evaluations, annotated datasets (e.g., CCGbank, Hockenmaier and Steedman, 2007)
- Learning for semantic parsing (Zettlemoyer and Collins, 2005) and CCG parsing (Clark and Curran, 2004a)
- Using CCG to represent other kinds of semantics (e.g., predicate-argument structures; Lewis and Steedman, 2014)
- Integrating continuous representations in semantic parsing (Lewis and Steedman, 2013; Krishnamurthy and Mitchell, 2013)
- Supertagging (Clark and Curran, 2004b) and making semantic parsing efficient (Lewis and Steedman, 2014)

To-Do List

- Compositional semantics chapter in Jurafsky and Martin (2008).

References I

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. In *Proc. of the Linguistic Annotation Workshop and Interoperability with Discourse*, 2013.

Bob Carpenter. *Type-logical semantics*. MIT Press, 1997.

Stephen Clark and James R. Curran. Parsing the WSJ using CCG and log-linear models. In *Proc. of ACL*, 2004a.

Stephen Clark and James R. Curran. The importance of supertagging for wide-coverage CCG parsing. In *Proc. of COLING*, 2004b.

Julia Hockenmaier and Mark Steedman. CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396, 2007.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, second edition, 2008.

Jayant Krishnamurthy and Tom Mitchell. Vector space semantic parsing: A framework for compositional vector space models. 2013.

Mike Lewis and Mark Steedman. Combining distributional and logical semantics. *Transactions of the Association for Computational Linguistics*, 1:179–192, 2013.

Mike Lewis and Mark Steedman. A* CCG parsing with a supertag-factored model. In *Proc. of EMNLP*, 2014.

References II

Richard Montague. Universal grammar. *Theoria*, 36:373–398, 1970.

Mark Steedman. *The Syntactic Process*. MIT Press, 2000.

Luke Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proc. of UAI*, 2005.