CS498JH: Introduction to NLP (Fall 2012)
http://cs.illinois.edu/class/cs498jh

Lecture 25: Compositional semantics

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center
Office Hours: Wednesday, 12:15-1:15pm

Semantics

In order to understand language, we need to know its meaning.

- What is the meaning of a word?
  (**Lexical semantics**)
- What is the meaning of a sentence?
  ([Compositional] semantics)
- What is the meaning of a longer piece of text?
  (Discourse semantics)

Why do we care about semantics?

Natural language conveys information about the world

We can compare statements about the world with the actual state of the world:
Champaign is in California. (false)

We can learn new facts about the world from natural language statements:
The earth turns around the sun.

We can answer questions about the world:
Where can I eat Korean food on campus?

We draw inferences from natural language statements

Some inferences are purely linguistic:

All blips are foos.
Blop is a blip.
Blop is a foo (whatever that is).

Some inferences require world knowledge.

Mozart was born in Salzburg.
Mozart was born in Vienna.
No, that can't be - these are different cities.

Today's lecture

Our initial question:
What is the meaning of (declarative) sentences?

Declarative sentences: *"John likes coffee"*.
(We won't deal with questions (*"Who likes coffee?"*) and imperative sentences (commands: *"Drink up!"*))

Follow-on question 1:
How can we represent the meaning of sentences?

Follow-on question 2:
How can we map a sentence to its meaning representation?

What do sentences mean?

Declarative sentences (statements) can be true or false, depending on the state of the world:

John sleeps.

In the simplest case, the consist of a verb and one or more noun phrase arguments.

Principle of compositionality (Frege):
The meaning of an expression depends on the meaning of its parts and how they are put together.

What do nouns and verbs mean?

In the simplest case, an NP is just a name: *John*
Names refer to entities in the world.

Verbs define n-ary predicates: depending on the arguments they take (and the state of the world), the result can be true or false.

First-order predicate logic (FOL)

FOL is sufficient for many Natural Language inferences

All blips are foos.	$\forall x\ blip(x) \rightarrow foo(x)$
Blop is a blip.	$blip(blop)$
Blop is a foo	$foo(blop)$

Some inferences require world knowledge.

Mozart was born in Salzburg.	$bornIn(Mozart, Salzburg)$
Mozart was born in Vienna.	$bornIn(Mozart, Vienna)$
No, that can't be- these are different cities	$bornIn(Mozart, Salzburg) \wedge \neg bornIn(Mozart, Salzburg)$

First-order predicate logic

Syntax: What is the language of well-formed formulas of predicate logic?

Semantics: What is the interpretation of a well-formed formula in predicate logic?
(This requires a model)

Inference rules and algorithms:
How can we reason with predicate logic?
(Not covered in this class)

Some examples

John is a student:
$student(john)$

All students take at least one class:
$\forall x\ student(x) \longrightarrow \exists y(class(y) \wedge takes(x,y))$

There is a class that all students take:
$\exists y(class(y) \wedge \forall x\ (student(x) \longrightarrow takes(x,y))$

Predicate logic expressions

Terms: refer to entities

Predicates: refer to relations or properties

Formulas: can be true or false

Terms

Terms refer to **entities** in the world

There are three kinds of terms:

Constants

Mary', John', Bevande', Urbana',…

Variables

x, y, z,…

***n*-ary functions applied to *n* terms:**

fatherOf(Mary'),

Predicates

Unary predicates define properties of entities:

student(john')

***N*-ary predicates** define relations between *n* entities:

fatherOf(john', tom')

Formulas

Atomic formulas are predicates, applied to terms:

book(x), eat(x,y)

Complex formulas are constructed recursively by

...**negation** (¬): *¬book(John')*

...**connectives** ($\wedge, \vee, \rightarrow$): *book(y)* $\wedge$ *read(x,y)*

conjunction (and): $\varphi \wedge \psi$ disjunction (or): $\varphi \vee \psi$ implication (if): $\varphi \rightarrow \psi$

...**quantifiers** ($\forall x$, $\exists x$)

universal (typically with implication) $\forall x[\varphi(x) \rightarrow \psi(x)]$

existential (typically with conjunction) $\exists x[\varphi(x)]$, $\exists x[\varphi(x) \wedge \psi(x)]$

Interpretation: formulas are either **true or false**.

The syntax of FOL expressions

Term ⇒ Constant |
Variable |
Function(Term,...,Term)

Formula ⇒ Predicate(Term, ...Term) |
¬ Formula |
∀ Variable Formula |
∃ Variable Formula |
Formula ∧ Formula |
Formula ∨ Formula |
Formula → Formula

Predicate logic models

Each model consist of a domain and an interpretation function of terms and predicates.

Domain: a set of entities: *{ann, peter, …,book1,…}*

Interpretation of terms: [[ann']] = ann

Unary predicates (properties) define (sub)sets of entities: *blue= {book25, sweater23, ...}*

N-ary predicates define sets of n-ary tuples of entities: *belongs_to = {<book25, peter>, <sweater23, ann>,…}*

Not all of natural language can be expressed in FOL:

Tense:
It was hot yesterday.
I will go to Chicago tomorrow.

Modals:
You can go to Chicago from here.

Other kinds of quantifiers:
Most students hate 8:00am lectures.

Using CCG to represent meaning

λ-Expressions

We often use **λ-expressions** to construct complex logical formulas:

- $\lambda x.\varphi(..x...)$ is a **function** where x is a variable, and φ some FOL expression.
- **β-reduction** (called λ-reduction in textbook): Apply $\lambda x.\varphi(..x...)$ to some argument a:
  $(\lambda x.\varphi(..x...)\ a) \Rightarrow \varphi(..a...)$
  Replace all occurrences of x in $\varphi(..x...)$ with a
- **n-ary functions** contain embedded λ-expressions:
  $\lambda x.\lambda y.\lambda z.give(x,y,z)$

Function application

Combines a function X/Y or X\Y with its argument Y to yield the result X:

(S\NP)/NP NP -> S\NP
eats tapas eats tapas

NP S\NP -> S
John eats tapas John eats tapas

Type-raising and composition

Type-raising: X → T/(T\X)

Turns an argument into a function.

NP → S/(S\NP) (subject)
NP → (S\NP)\((S\NP)/NP) (object)

Harmonic composition: X/Y Y/Z → X/Z

Composes two functions (complex categories)

(S\NP)/PP PP/NP → (S\NP)/NP
S/(S\NP) (S\NP)/NP → S/NP

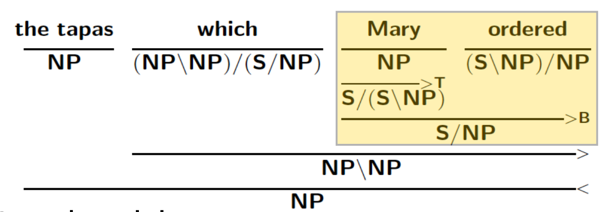
Crossing function composition: X/Y Y\Z → X\Z

Composes two functions (complex categories)

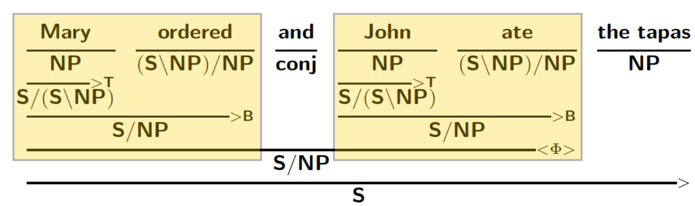
(S\NP)/S S\NP → (S\NP)\NP

Type-raising and composition

Wh-movement (relative clause):



Right-node raising:



An example

$$\dfrac{\dfrac{John}{NP} \quad \dfrac{\dfrac{sees}{(S\backslash NP)/NP} \quad \dfrac{Mary}{NP}}{S\backslash NP}>}{S}<$$

CCG semantics

Every syntactic constituent has a semantic interpretation:

Every **lexical entry** maps a word to a syntactic category and a corresponding semantic type:

*John=(**NP**, john') Mary= (**NP**, mary')*
*loves: ((**S\NP)/NP** λx.λy.loves(x,y))*

Every **combinatory rule** has a syntactic and a semantic part:

Function application: $X/Y{:}\lambda x.f(x)\;\; Y{:}a \rightarrow X{:}f(a)$
Function composition: $X/Y{:}\lambda x.f(x)\;\; Y/Z{:}\lambda y.g(y) \rightarrow X/Z{:}\lambda z.f(\lambda y.g(y).z)$
Type raising: $X{:}a \rightarrow T/(T\backslash X)\; \lambda f.f(a)$

An example with semantics

$$\dfrac{\dfrac{John}{NP : John} \quad \dfrac{\dfrac{sees}{(S\backslash NP)/NP : \lambda x.\lambda y.sees(x,y)} \quad \dfrac{Mary}{NP : Mary}}{S\backslash NP : \lambda y.sees(Mary,y)}>}{S : sees(Mary, John)}<$$

Quantifier scope ambiguity

"Every chef cooks a meal"

- **Interpretation A:**
  For every chef, there is a meal which he cooks.

$$\forall x[chef(x) \rightarrow \exists y[meal(y) \wedge cooks(y,x)]]$$

- **Interpretation B:**
  There is some meal which every chef cooks.

$$\exists y[meal(y) \wedge \forall x[chef(x) \rightarrow cooks(y,x)]]$$

Supplementary material: quantifier scope ambiguities in CCG

Interpretation A

Every	chef	cooks	a	meal
$(S/(S\backslash NP))/N$	N	$(S\backslash NP)/NP$	$((S\backslash NP)\backslash((S\backslash NP)/NP))/N$	N
$\lambda P\lambda Q.\forall x[Px \rightarrow Qx]$	$\lambda z.chef(z)$	$\lambda u.\lambda v.cooks(u,v)$	$\lambda P\lambda Q\exists y[Py \wedge Qy]$	$\lambda z.meal(z)$

Interpretation A

Every	chef	cooks	a	meal
$(S/(S\backslash NP))/N$	N	$(S\backslash NP)/NP$	$((S\backslash NP)\backslash((S\backslash NP)/NP))/N$	N
$\lambda P\lambda Q.\forall x[Px \rightarrow Qx]$	$\lambda z.chef(z)$	$\lambda u.\lambda v.cooks(u,v)$	$\lambda P\lambda Q\exists y[Py \wedge Qy]$	$\lambda z.meal(z)$

Every chef $>$:
$$S/(S\backslash NP)$$
$$\lambda Q.\forall x[\lambda z.chef(z)x \rightarrow Qx] \equiv \lambda Q.\forall x[chef(x) \rightarrow Qx]$$

a meal $>$:
$$(S\backslash NP)\backslash((S\backslash NP)/NP)$$
$$\lambda Q\exists y[\lambda z.meal(z)y \wedge Qy] \equiv \lambda Q\lambda w.\exists y[meal(y) \wedge Qyw]$$

Interpretation A

Every	chef	cooks	a	meal
$(S/(S\backslash NP))/N$	N	$(S\backslash NP)/NP$	$((S\backslash NP)\backslash((S\backslash NP)/NP))/N$	N
$\lambda P\lambda Q.\forall x[Px \rightarrow Qx]$	$\lambda z.chef(z)$	$\lambda u.\lambda v.cooks(u,v)$	$\lambda P\lambda Q\exists y[Py \wedge Qy]$	$\lambda z.meal(z)$

Every chef $>$:
$$S/(S\backslash NP)$$
$$\lambda Q.\forall x[\lambda z.chef(z)x \rightarrow Qx] \equiv \lambda Q.\forall x[chef(x) \rightarrow Qx]$$

a meal $>$:
$$(S\backslash NP)\backslash((S\backslash NP)/NP)$$
$$\lambda Q\exists y[\lambda z.meal(z)y \wedge Qy] \equiv \lambda Q\lambda w.\exists y[meal(y) \wedge Qyw]$$

cooks a meal $<$:
$$S\backslash NP$$
$$\lambda w.\exists y[meal(y) \wedge \lambda u\lambda v.cooks(u,v)yw] \equiv \lambda w.\exists y[meal(y) \wedge cooks(y,w)]$$

Interpretation A

Every	chef	cooks	a	meal
$(S/(S\backslash NP))/N$	N	$(S\backslash NP)/NP$	$((S\backslash NP)\backslash((S\backslash NP)/NP))/N$	N
$\lambda P\lambda Q.\forall x[Px \rightarrow Qx]$	$\lambda z.chef(z)$	$\lambda u.\lambda v.cooks(u,v)$	$\lambda P\lambda Q\exists y[Py \land Qy]$	$\lambda z.meal(z)$

Every chef ($>$):
$$S/(S\backslash NP)$$
$$\lambda Q.\forall x[\lambda z.chef(z)x \rightarrow Qx] \equiv \lambda Q.\forall x[chef(x) \rightarrow Qx]$$

a meal ($>$):
$$(S\backslash NP)\backslash((S\backslash NP)/NP)$$
$$\lambda Q\exists y[\lambda z.meal(z)y \land Qy] \equiv \lambda Q\lambda w.\exists y[meal(y) \land Qyw]$$

cooks a meal ($<$):
$$S\backslash NP$$
$$\lambda w.\exists y[meal(y) \land \lambda u\lambda v.cooks(u,v)yw] \equiv \lambda w.\exists y[meal(y) \land cooks(y,w)]$$

Every chef cooks a meal ($>$):
$$S : \forall x[chef(x) \rightarrow \lambda w.\exists y[meal(y) \land cooks(y,w)]x] \equiv \forall x[chef(x) \rightarrow \exists y[meal(y) \land cooks(y,x)]]$$

Interpretation B

Every	chef	cooks	a	meal
$(S/(S\backslash NP))/N$	N	$(S\backslash NP)/NP$	$(S\backslash(S/NP))/N$	N
$\lambda P\lambda Q.\forall x[Px \rightarrow Qx]$	$\lambda z.chef(z)$	$\lambda u.\lambda v.cooks(u,v)$	$\lambda P\lambda Q\exists y[Py \land Qy]$	$\lambda z.meal(z)$

Interpretation B

Every	chef	cooks	a	meal
$(S/(S\backslash NP))/N$	N	$(S\backslash NP)/NP$	$(S\backslash(S/NP))/N$	N
$\lambda P\lambda Q.\forall x[Px \rightarrow Qx]$	$\lambda z.chef(z)$	$\lambda u.\lambda v.cooks(u,v)$	$\lambda P\lambda Q\exists y[Py \land Qy]$	$\lambda z.meal(z)$

Every chef ($>$):
$$S/(S\backslash NP)$$
$$\lambda Q\forall x[\lambda z.chef(z)x \rightarrow Qx] \equiv \lambda Q\forall x[chef(x) \rightarrow Qx]$$

a meal ($>$):
$$S\backslash(S/NP)$$
$$\lambda Q\exists y[\lambda z.meal(z)y \land Qy] \equiv \lambda Q\exists y[meal(y) \land Qy]$$

Interpretation B

Every	chef	cooks	a	meal
$(S/(S\backslash NP))/N$	N	$(S\backslash NP)/NP$	$(S\backslash(S/NP))/N$	N
$\lambda P\lambda Q.\forall x[Px \rightarrow Qx]$	$\lambda z.chef(z)$	$\lambda u.\lambda v.cooks(u,v)$	$\lambda P\lambda Q\exists y[Py \land Qy]$	$\lambda z.meal(z)$

Every chef ($>$):
$$S/(S\backslash NP)$$
$$\lambda Q\forall x[\lambda z.chef(z)x \rightarrow Qx] \equiv \lambda Q\forall x[chef(x) \rightarrow Qx]$$

a meal ($>$):
$$S\backslash(S/NP)$$
$$\lambda Q\exists y[\lambda z.meal(z)y \land Qy] \equiv \lambda Q\exists y[meal(y) \land Qy]$$

Every chef cooks ($>B$):
$$S/NP$$
$$\lambda w.\forall x[chef(x) \rightarrow \lambda u\lambda v.cooks(u,v)wx] \equiv \lambda w.\forall x[chef(x) \rightarrow cooks(w,x)]$$

Interpretation B

Every	chef	cooks	a	meal
$(S/(S\backslash NP))/N$	N	$(S\backslash NP)/NP$	$(S\backslash(S/NP))/N$	N
$\lambda P\lambda Q.\forall x[Px \rightarrow Qx]$	$\lambda z.chef(z)$	$\lambda u.\lambda v.cooks(u,v)$	$\lambda P\lambda Q\exists y[Py \wedge Qy]$	$\lambda z.meal(z)$

Every chef ($>$):

$$S/(S\backslash NP) \quad \lambda Q\forall x[\lambda z.chef(z)x \rightarrow Qx] \equiv \lambda Q\forall x[chef(x) \rightarrow Qx]$$

a meal ($>$):

$$S\backslash(S/NP) \quad \lambda Q\exists y[\lambda z.meal(z)y \wedge Qy] \equiv \lambda Q\exists y[meal(y) \wedge Qy]$$

Every chef cooks ($>B$):

$$S/NP \quad \lambda w.\forall x[chef(x) \rightarrow \lambda u\lambda v.cooks(u,v)wx] \equiv \lambda w.\forall x[chef(x) \rightarrow cooks(w,x)]$$

Every chef cooks a meal ($<$):

$$S \exists y[meal(y) \wedge \lambda w.\forall x[chef(x) \rightarrow cooks(y,w)]x] \equiv \exists y[meal(y) \wedge \forall x[chef(x) \rightarrow cooks(y,x)]]$$

Additional topics

Representing events and temporal relations:

- Add event variables *e* to represent the events described by verbs, and temporal variables *t* to represent the time at which an event happens.

Other quantifiers:

- What about "*most / at least two / … chefs*"?

Underspecified representations:

- Which interpretation of *"Every chef cooks a meal"* is correct? This might depend on context. Let the parser generate an underspecified representation from which both readings can be computed.

Going beyond single sentences:

- How do we combine the interpretations of single sentences?

Today's key concepts

Why do we need to represent meaning?

Inference

Interactions with (general) world knowledge and situational context

How do we represent meaning?

First order predicate logic

Semantics in CCG

Today's reading

Textbook:

Chapter 17, sections 1-3

Chapter 18, section 2 (for a slightly different treatment of computational semantics)

Optional: Chapter 18, section 3 (underspecified representations)

Additional material

The interpretation of FOL expressions

A model is a pair $M=(D,I)$ where:

- The **domain** D is a **nonempty set of objects**
  e.g. $D = \{a12, b45, c843, ...\}$

- The **interpretation function** I maps:
  each **constant** c to an element c^I of D,
  each *n*-place **function** symbol f to an *n*-ary function $f^I: D_n \rightarrow D$,
  and each *n*-place **predicate** symbol p to an n-ary relation $p^I: D^n \rightarrow \{true, false\}$
  e.g. $John^I = a12$, $fatherOf^I(a12) = b45$,
  $child^I = \{a12, a14, ...\}$, $likes^I = \{\langle a12, b45\rangle, \langle b45, a12\rangle ...\}$

The interpretation of FOL expressions (2)

- A **variable assignment** g over a domain D is a function from the set of variables to the set of elements of D.

- The **valuation function** $val_{I,g}$ over terms is defined recursively as:
  $val_{I,g}(x) = g(x)$ for variables
  $val_{I,g}(c) = c^I$ for constants
  $val_{I,g}(f(t_1, ..., t_n)) = f^I(val_{I,g}(t_1), ..., val_{I,g}(t_n))$ for functions

- The **substitution** $[u/x]$ replaces variable x with some element u of D.

The satisfaction relation

- We write $M,g \models \varphi$ to say that the formula φ is **satisfied** in the model $M=(D,I)$ under assignment g

- The relation $\models$ is defined as follows:
  $M,g \models P(t_1,...t_n)$ iff $P^I(val_{I,g}(t_1),... val_{I,g}(t_n)) = true$
  $M,g \models \neg\varphi$ iff not $M,g \models \varphi$
  $M,g \models \varphi \wedge \psi$ iff $M,g \models \varphi$ and $M,g \models \psi$
  $M,g \models \varphi \vee \psi$ iff $M,g \models \varphi$ or $M,g \models \psi$
  $M,g \models \varphi \rightarrow \psi$ iff $M,g \models \neg\varphi$ or $M,g \models \psi$
  $M,g \models \forall x\varphi$ iff $M,g \models \varphi[u/x]$ for all substitutions u for x.
  $M,g \models \exists x\varphi$ iff $M,g \models \varphi[u/x]$ for some substitution u for x