Introduction to semantic parsing and the lambda calculus

Bill MacCartney

CS224U

28 April 2014

Reminder

Lit Review due in one week!

Time to get cracking!

Full understanding?

- We're doing natural language *understanding*, right?
- Are we there yet? Do we fully *understand*?
    - With VSMs? Dependency parses? Relation extraction?
    - Arguably, all are steps toward to NLU … but are they sufficient?
- What aspects of meaning are we still unable to capture?
    - Higher-arity relations, events with multiple participants, temporal aspects, negation, disjunction, quantification, propositional attitudes, modals, ...

Logic games from LSAT (& old GRE)

Six sculptures — C, D, E, F, G, H — are to be exhibited in rooms 1, 2, and 3 of an art gallery.

- Sculptures C and E may not be exhibited in the same room.
- Sculptures D and G must be exhibited in the same room.
- If sculptures E and F are exhibited in the same room, no other sculpture may be exhibited in that room.
- At least one sculpture must be exhibited in each room, and no more than three sculptures may be exhibited in any room.

If sculpture D is exhibited in room 3 and sculptures E and F are exhibited in room 1, which of the following may be true?

A. Sculpture C is exhibited in room 1.
B. Sculpture H is exhibited in room 1.
C. Sculpture G is exhibited in room 2.
D. Sculptures C and H are exhibited in the same room.
E. Sculptures G and F are exhibited in the same room.

Travel reservations

Yes, hi, I need to book a flight for myself and my husband from Boston to SFO, or Oakland would be OK too. We need to fly out on Friday the 12th, and then I could come back on Sunday evening or Monday morning, but he won't return until Wednesday the 18th, because he's staying for business. No flights with more than one stop, and we don't want to fly on United because we hate their guts.

SHRDLU (Winograd 1972)

Find a block which is taller than the one you are holding and put it into the box.

OK.

How many blocks are not in the box?

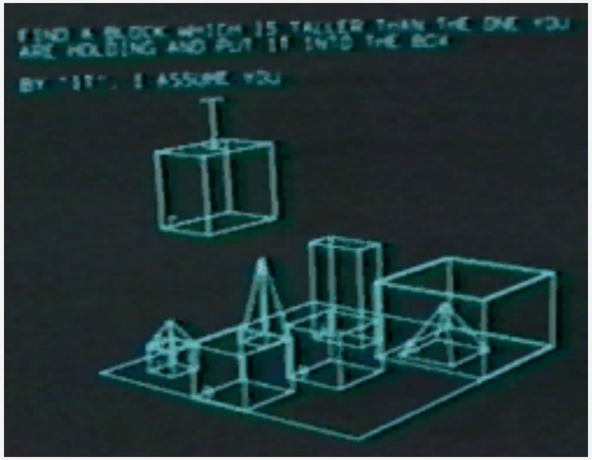
FOUR OF THEM.

Is at least one of them narrower than the one which I told you to pick up?

YES, THE RED CUBE.

http://youtube.com/watch?v=8SvD-lNg0TA

http://hci.stanford.edu/winograd/shrdlu/



CHAT-80

- Developed 1979-82 by Fernando Pereira & David Warren
- Proof-of-concept natural language interface to database
- Could answer questions about geography
- Implemented in Prolog
- Hand-built lexicon & grammar
- Highly influential NLIDB system

CHAT-80 demo

You can run Chat-80 yourself on the myth machines!

1. `ssh myth.stanford.edu`
2. `cd /afs/ir/class/cs224n/src/chat/`
3. `/usr/sweet/bin/sicstus`
4. `[load].`
5. `hi.`
6. `what is the capital of france?`

Sample queries can be found at:

`/afs/ir/class/cs224n/src/chat/demo`

All the source code is there for your perusal as well

Things you could ask CHAT-80

- Is there more than one country in each continent?
- What countries border Denmark?
- What are the countries from which a river flows into the Black_Sea?
- What is the total area of countries south of the Equator and not in Australasia?
- Which country bordering the Mediterranean borders a country that is bordered by a country whose population exceeds the population of India?
- How far is London from Paris? I don’t understand!

The CHAT-80 database

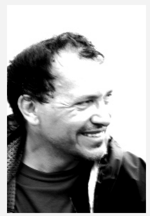
```
% Facts about countries.
% country(Country, Region, Latitude, Longitude,
%   Area(sqmiles), Population, Capital, Currency)
country(andorra, southern_europe, 42, -1, 179, 25000,
andorra_la_villa, franc_peseta).
country(angola, southern_africa, -12, -18, 481351,
5810000, luanda, ?).
country(argentina, south_america, -35, 66, 1072067,
23920000, buenos_aires, peso).

capital(C,Cap) :- country(C,_,_,_,_,_,Cap,_).
```

The CHAT-80 grammar

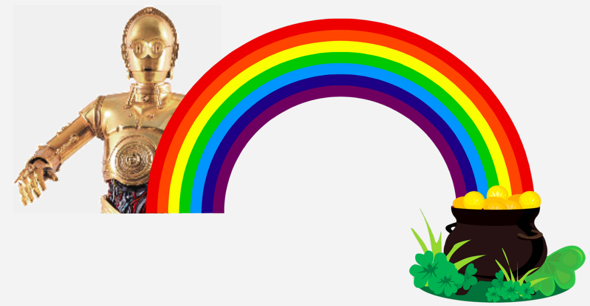
```
/* Sentences */
sentence(S) --> declarative(S), terminator(.) .
sentence(S) --> wh_question(S), terminator(?) .
sentence(S) --> yn_question(S), terminator(?) .
sentence(S) --> imperative(S), terminator(!) .

/* Noun Phrase */
np(np(Agmt,Pronoun,[]),Agmt,NPCase,def,_,Set,Nil) -->
  {is_pp(Set)},
  pers_pron(Pronoun,Agmt,Case),
  {empty(Nil), role(Case,decl,NPCase)}.

/* Prepositional Phrase */
pp(pp(Prep,Arg),Case,Set,Mask) -->
  prep(Prep),
  {prep_case(NPCase)},
  np(Arg,_,NPCase,_,Case,Set,Mask).
```

Precision vs. robustness

Precise, complete understanding

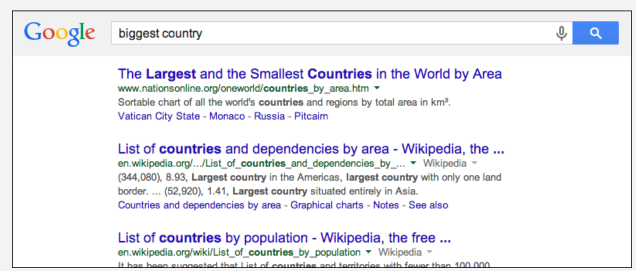


SHRDLU CHAT-80



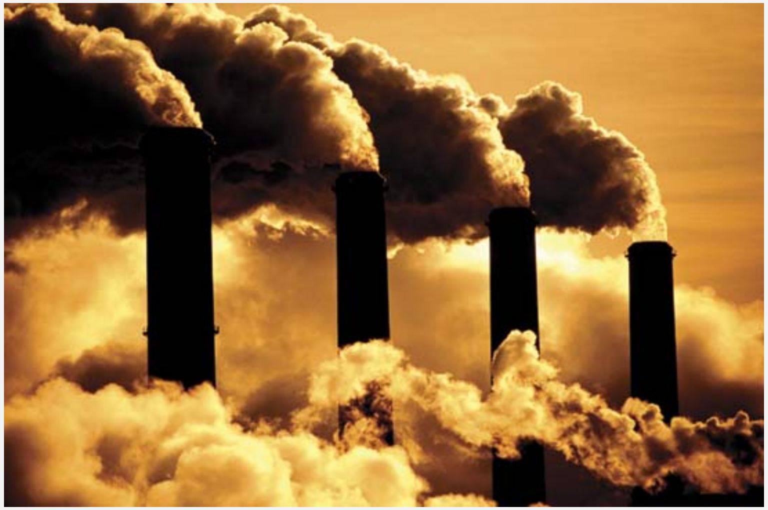
Brittle, narrow coverage

Robust, broad coverage



Fuzzy, partial understanding

Carbon emissions



Which country has the highest CO2 emissions?
What about highest per capita?
Which had the biggest increase over the last five years?
What fraction was from European countries?

Baseball statistics

Pitchers who have struck out four batters in one inning
Players who have stolen at least 100 bases in a season
Complete games with fewer than 90 pitches
Most home runs hit in one game

Voice commands

How do I get to the Ferry Building by bike
Book a table for four at Nopa on Friday after 9pm
Text my wife I'm going to be twenty minutes late
Add House of Cards to my Netflix queue at the top

Semantic parsing

If we want to understand natural language *completely* and *precisely*, we need to do semantic parsing.

That is, translate natural language into a formal meaning representation on which a machine can act.

First, we need to define our goal.

What should we choose as our target output representation of meaning?

Database queries

To facilitate data exploration and analysis, you might want to parse natural language into database queries:

which country had the highest carbon emissions last year

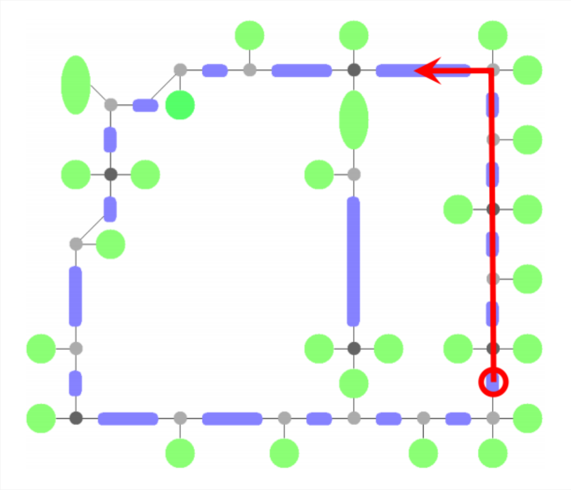
```
SELECT    country.name
FROM      country, co2_emissions
WHERE     country.id = co2_emissions.country_id
AND       co2_emissions.year = 2013
ORDER BY co2_emissions.volume DESC
LIMIT     1;
```

Robot control

For a robot control application, you might want a custom-designed procedural language:

Go to the third junction and take a left.

```
(do-sequentially
  (do-n-times 3
    (do-sequentially
      (move-to forward-loc)
      (do-until
        (junction current-loc)
        (move-to forward-loc))))
  (turn-left))
```



Intents and arguments

For smartphone voice commands, you might want relatively simple meaning representations, with *intents* and *arguments*:

directions to SF by train

```
(TravelQuery
  (Destination /m/0d6lp)
  (Mode TRANSIT))
```

angelina jolie net worth

```
(FactoidQuery
  (Entity /m/0f4vbz)
  (Attribute /person/net_worth))
```

weather friday austin tx

```
(WeatherQuery
  (Location /m/0vzm)
  (Date 2013-12-13))
```

text my wife on my way

```
(SendMessage
  (Recipient 0x31cbf492)
  (MessageType SMS)
  (Subject "on my way"))
```

play sunny by boney m

```
(PlayMedia
  (MediaType MUSIC)
  (SongTitle "sunny")
  (MusicArtist /m/017mh))
```

is REI open on sunday

```
(LocalQuery
  (QueryType OPENING_HOURS)
  (Location /m/02nx4d)
  (Date 2013-12-15))
```

Demo: wit.ai

For a very simple NLU system based on identifying intents and arguments, check out this startup:

http://wit.ai/

First-order logic

Blackburn & Bos make a strong argument for using first-order logic as the meaning representation.

Powerful, flexible, general.

Can subsume most other representations as special cases.

John walks	`walk(john)`
John loves Mary	`love(john, mary)`
Every man loves Mary	$\forall$`x (man(x) → love(x, mary))`

(Lambda calculus will be the vehicle; first-order logic will be the final destination.)

FOL syntax, in a nutshell

- FOL symbols
  - Constants: john, mary
  - Predicates & relations: man, walks, loves
  - Variables: x, y
  - Logical connectives: $\wedge$ $\vee$ $\neg$ $\rightarrow$
  - Quantifiers: $\forall$ $\exists$
  - Other punctuation: parens, commas

"content words" (user-defined)

"function words"

- FOL formulae
  - Atomic formulae: loves(john, mary)
  - Connective applications: man(john) $\wedge$ loves(john, mary)
  - Quantified formulae: $\exists x\,(\text{man}(x))$

An NLU pipeline

- English sentences

  John smokes. Everyone who smokes snores.

- Syntactic analysis

  (S (NP John) (VP smokes))

- Semantic analysis

  smoke(john)

Focus of semantic parsing

- Inference

  $\forall$x.smoke(x) $\rightarrow$ snore(x), smoke(john)

  $\Rightarrow$ snore(john)

From language to logic

How can we design a general algorithm for translating from natural language into logical formulae?

John walks	walk(john)
John loves Mary	love(john, mary)
A man walks	$\exists$x.man(x) $\wedge$ walk(x)
A man loves Mary	$\exists$x.man(x) $\wedge$ love(x, mary)
John and Mary walk	walk(john) $\wedge$ walk(mary)
Every man walks	$\forall$x.man(x) $\rightarrow$ walk(x)
Every man loves a woman	$\forall$x.man(x) $\rightarrow$ $\exists$y.woman(y) $\wedge$ love(x, y)

We don't want to simply memorize these pairs, because that won't generalize to new sentences.

Machine translation (MT)

How can we design a general algorithm for translating from one language into another?

John walks	Jean marche
John loves Mary	Jean aime Marie
A man walks	Un homme marche
A man loves Mary	Un homme aime Marie
John and Mary walk	Jean et Marie marche
Every man walks	Chaque homme marche
Every man loves a woman	Chaque homme aime une femme

In MT, we break the input into pieces, translate the pieces, and then put the pieces back together.

A logical lexicon (first attempt)

John walks	walk(john)
John loves Mary	love(john, mary)
A man walks	$\exists$x.man(x) $\wedge$ walk(x)
A man loves Mary	$\exists$x.man(x) $\wedge$ love(x, mary)
John and Mary walk	walk(john) $\wedge$ walk(mary)
Every man walks	$\forall$x.man(x) $\rightarrow$ walk(x)
Every man loves a woman	$\forall$x.man(x) $\rightarrow$ $\exists$y.woman(y) $\wedge$ love(x, y)

John : john
Mary : mary

walks : walk(?)
loves : love(?, ?)

and : $\wedge$

man : man(?)
woman : woman(?)

a : $\exists$x.? $\wedge$?
every : $\forall$x.? $\rightarrow$?

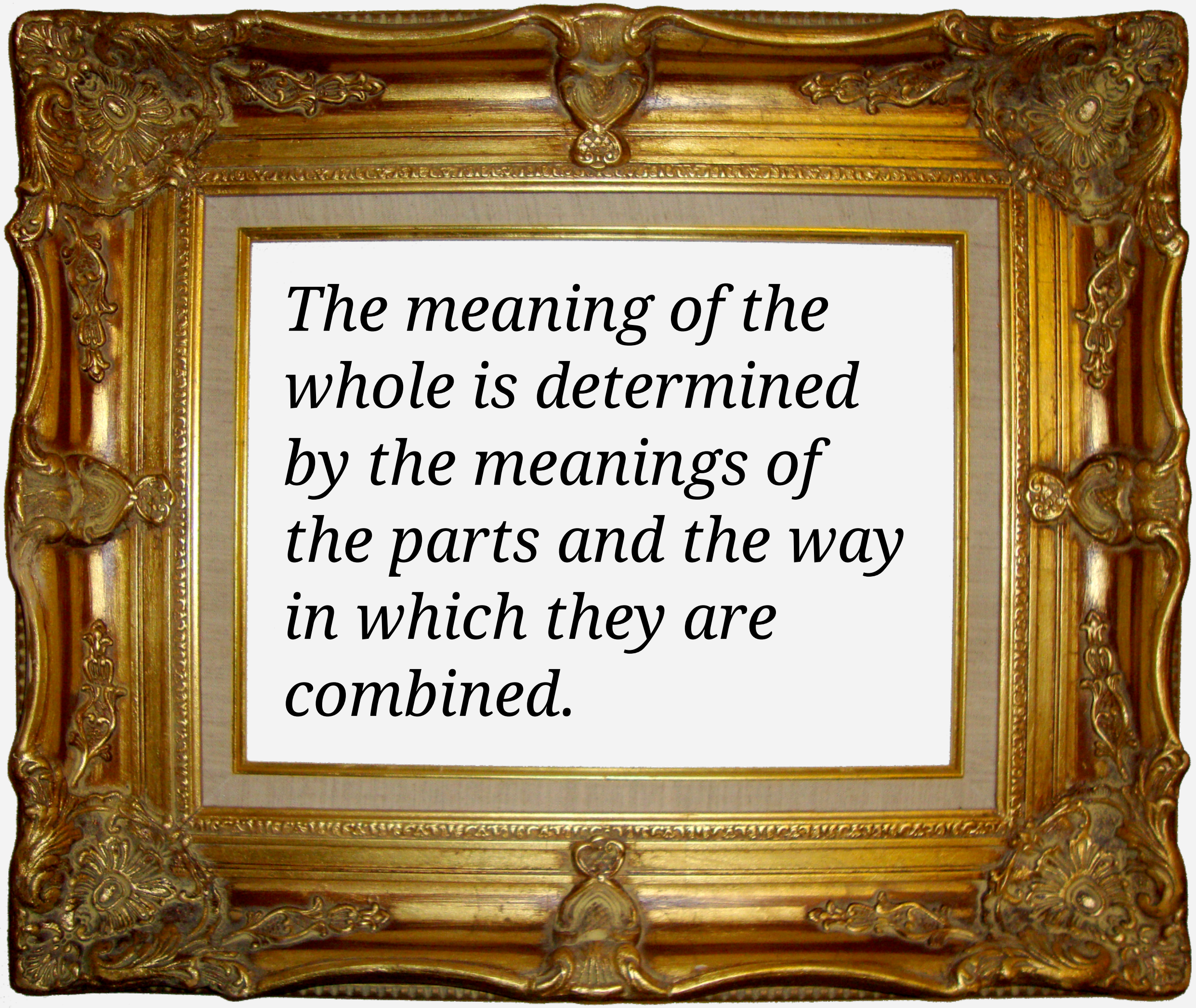
Compositional semantics

Now how do we put the pieces back together?

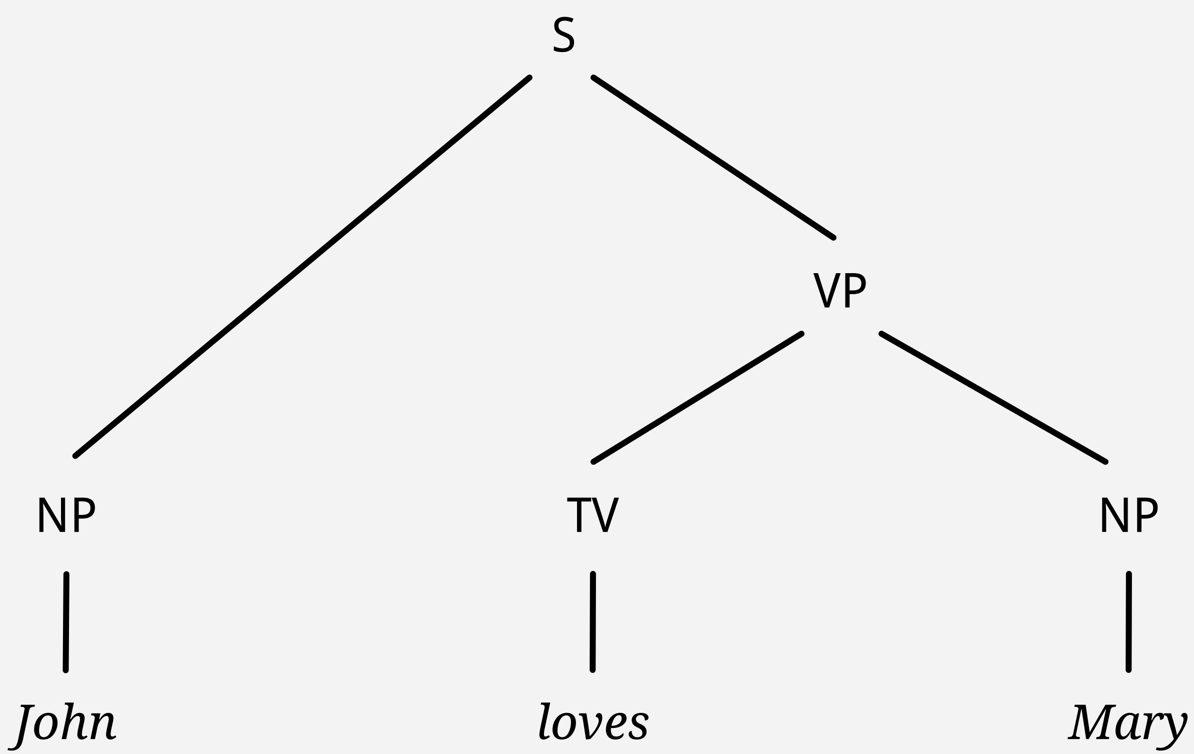
Idea: syntax-driven compositional semantics

1. Parse sentence to get syntax tree
2. Look up the semantics of each word in lexicon
3. Build the semantics for each constituent bottom-up, by combining the semantics of its children

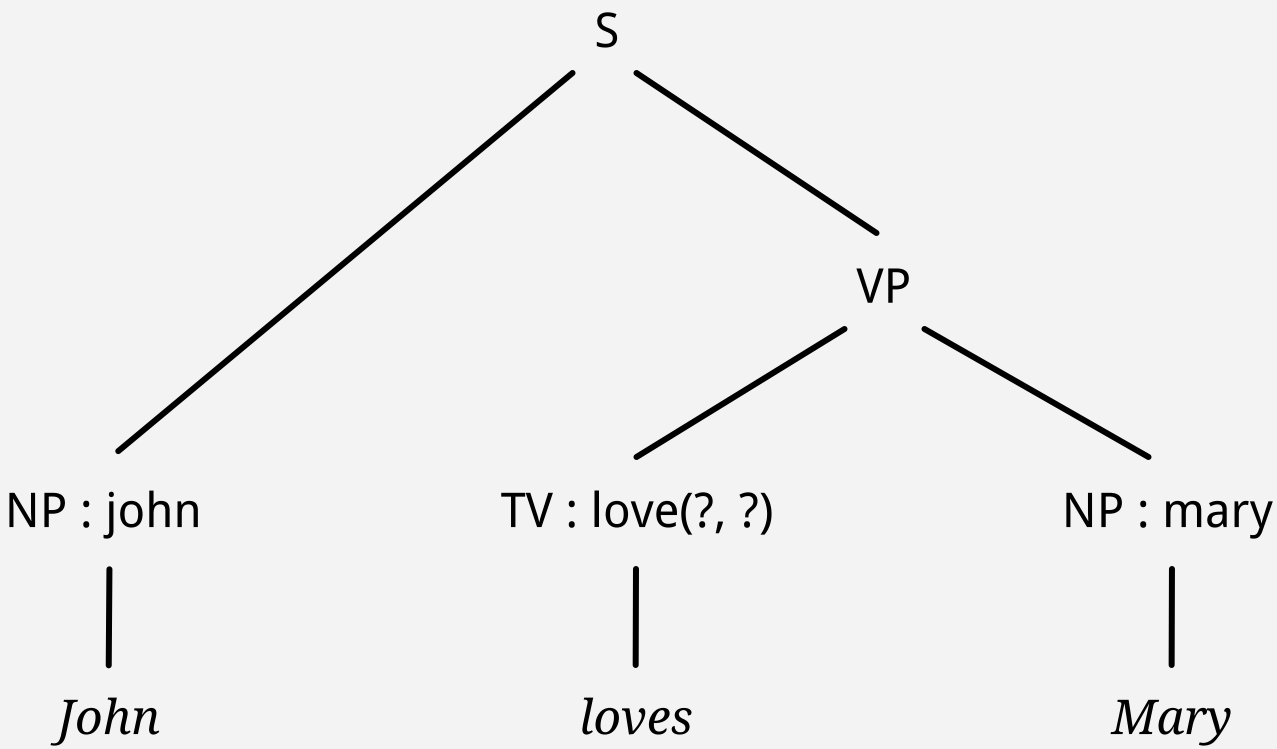
Principle of compositionality



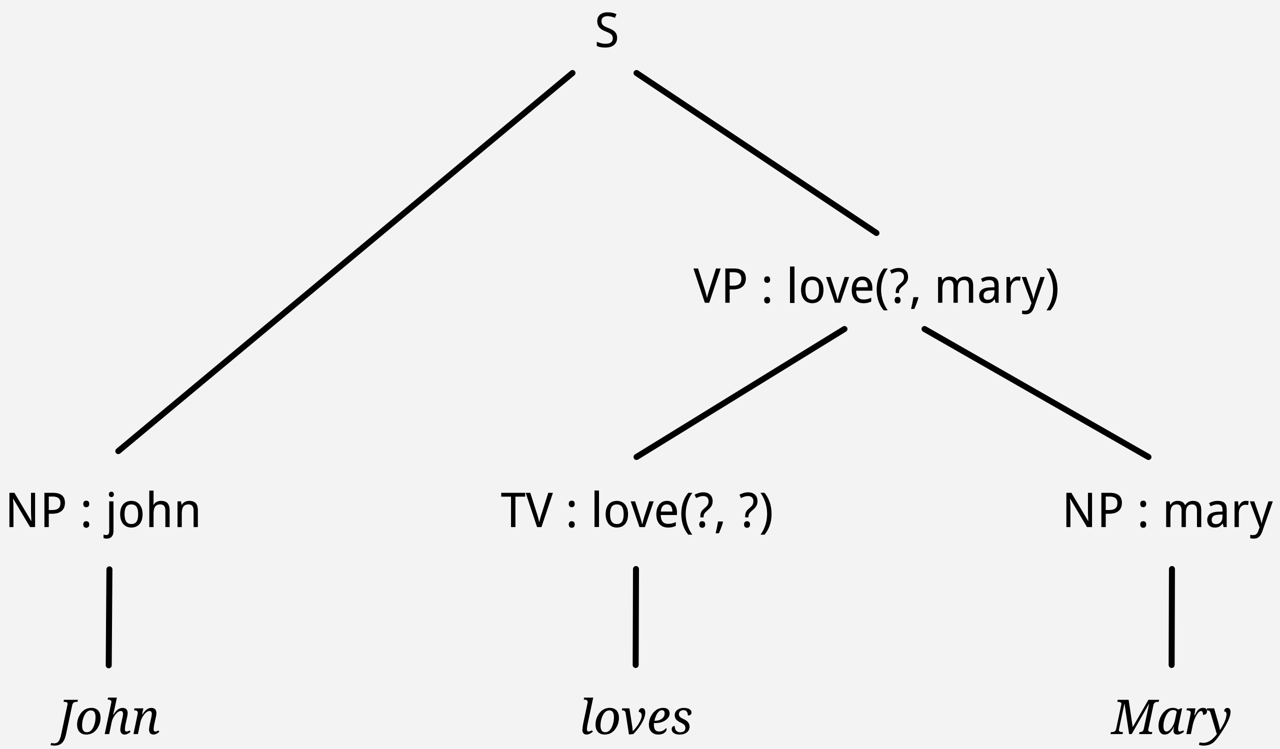
Example: syntactic analysis



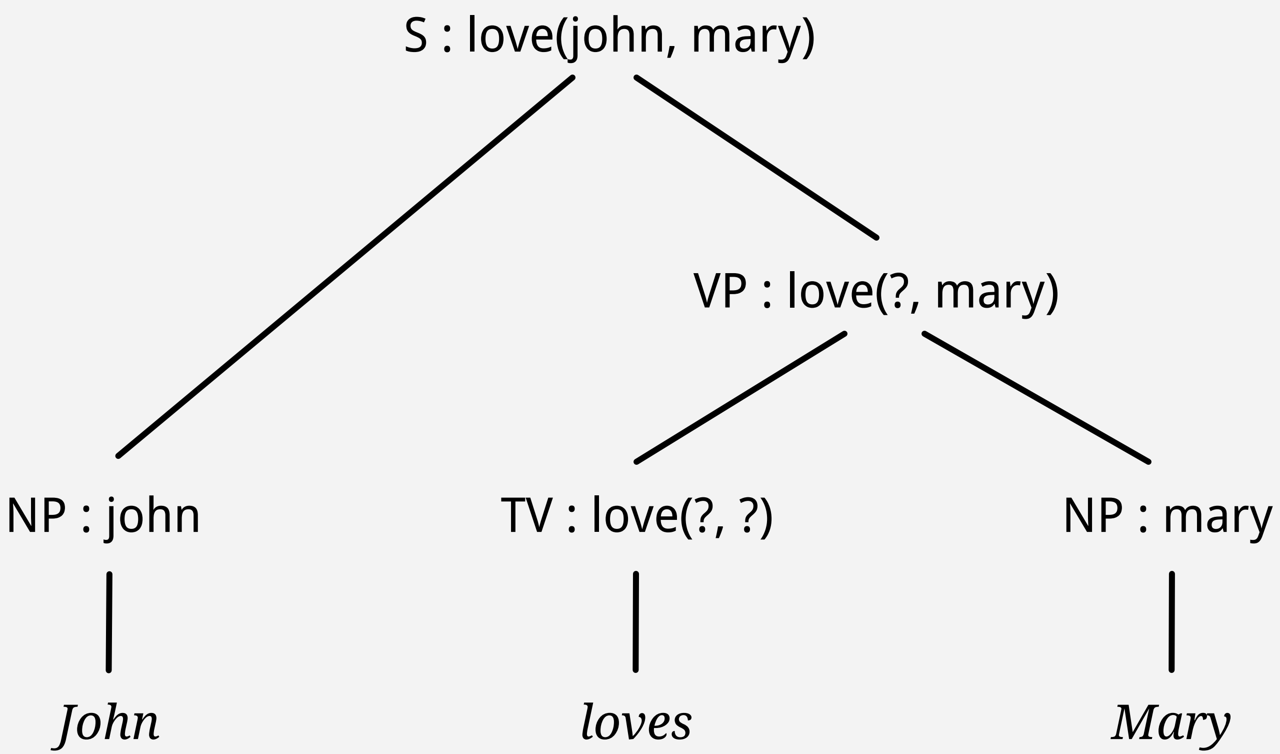
Example: semantic lexicon



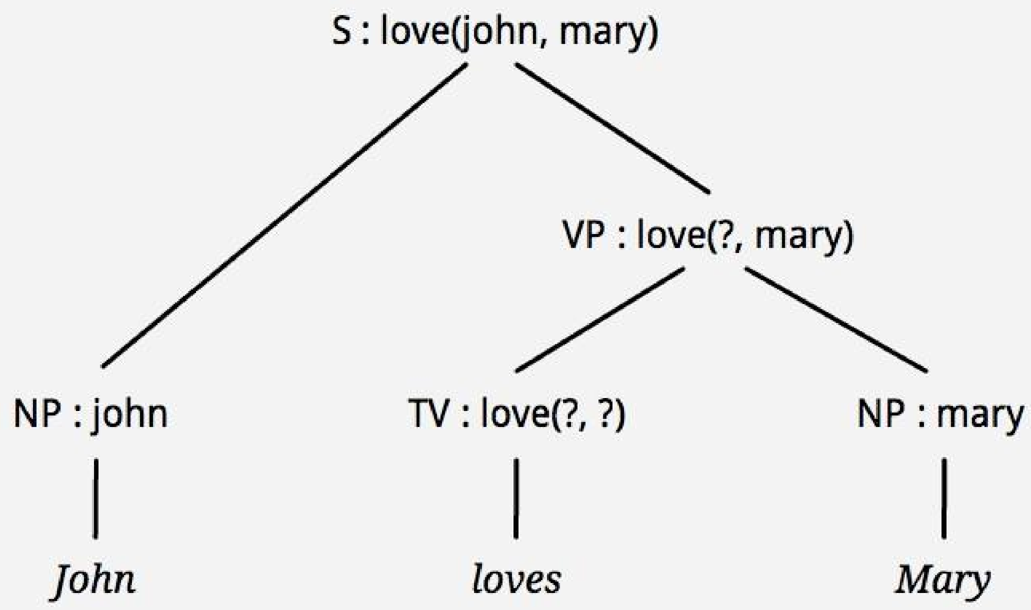
Example: semantic composition



Example: semantic composition



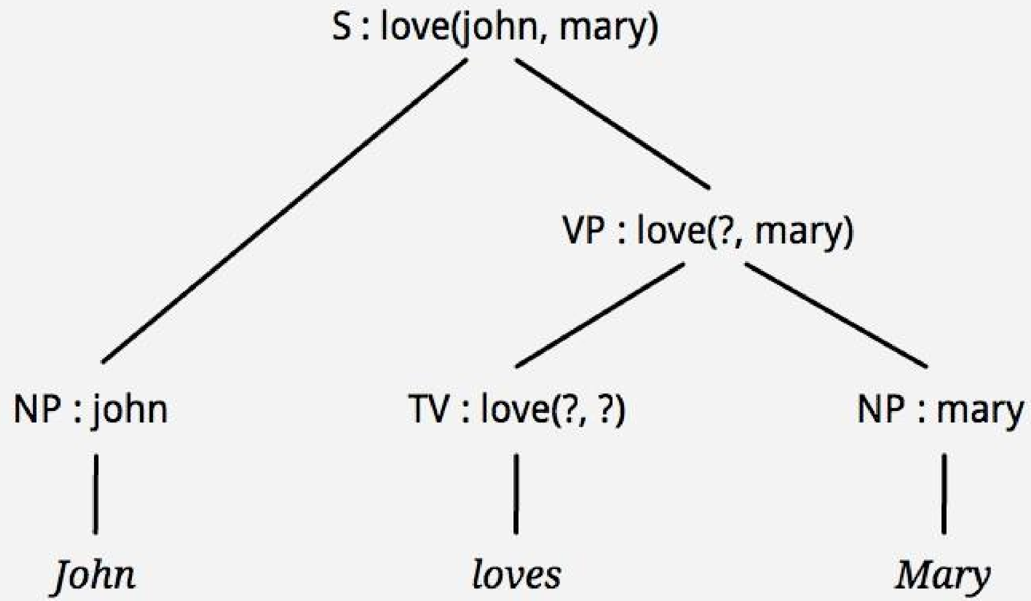
Compositionality



The meaning of the sentence is constructed from:

- the meaning of the words (i.e., the lexicon)
- paralleling the syntactic construction (i.e., the semantic rules)

Systematicity



How do we know how to construct the VP?

love(?, mary) OR love(mary, ?)

How can we *specify* in which way the bits & pieces combine?

Systematicity (continued)

- How do we want to represents parts of formulae?

  E.g. for the VP *loves Mary* ?

  love(?, mary)   bad: not FOL

  love(x, mary)   bad: no control over free variable

- Familiar well-formed formulae (sentences)

  $\forall$x (love(x, mary))   *Everyone loves Mary*

  $\exists$x (love(mary, x))   *Mary loves someone*

Lambda abstraction

- Add a new operator λ to bind free variables

  λx.love(x, mary)   *loves Mary*

- The new meta-logical symbol λ marks missing information in the object language (λ-)FOL

- We *abstract* over x

- Just like in programming languages!

  Python: `lambda x: x % 2 == 0`

  Ruby: `lambda {|x| x % 2 == 0}`

- How do we combine these new formulae and terms?

Super glue

- Gluing together formulae/terms with function application

  (λx.love(x, mary)) @ john

  (λx.love(x, mary))(john)

- How do we get back to the familiar love(john, mary) ?

- FA triggers a simple operation: *beta reduction*

  replace the λ-bound variable by the argument throughout the body

Beta reduction

$(\lambda x.love(x, mary))\ (john)$

1. Strip off the λ prefix

$(love(x, mary))\ (john)$

2. Remove the argument

$love(x, mary)$

3. Replace all occurrences of λ-bound variable by argument

$love(john, mary)$

Application vs. abstraction

$(\lambda x.\text{love}(x, \text{mary}))\ (\text{john})$

application (β-reduction)

abstraction

$\text{love}(\text{john}, \text{mary})$

Semantic construction with lambdas

S : $(\lambda x.\text{love}(x, \text{mary}))(\text{john})$
$= \text{love}(\text{john}, \text{mary})$

VP : $(\lambda y.\lambda x.\text{love}(x, y))(\text{mary})$
$= \lambda x.\text{love}(x, \text{mary})$

NP : john

TV : $\lambda y.\lambda x.\text{love}(x, y)$

NP : mary

John

loves

Mary

A semantic grammar

Lexicon

John	$\leftarrow$	NP : john
Mary	$\leftarrow$	NP : mary
loves	$\leftarrow$	TV : $\lambda y.\lambda x.\mathrm{love}(x, y)$

Composition rules

$$\mathrm{VP} : f(a) \rightarrow \mathrm{TV} : f \quad \mathrm{NP} : a$$

$$\mathrm{S} : f(a) \rightarrow \mathrm{NP} : a \quad \mathrm{VP} : f$$

Note the *semantic attachments* — these are augmented CFG rules
Note the use of function application to glue things together
For binary rules, four possibilities for semantics of parent (what?)

Montague semantics

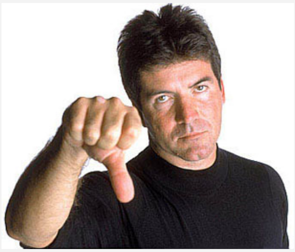
This approach to formal semantics was pioneered by Richard Montague (1930-1971)

"… I reject the contention that an important theoretical difference exists between formal and natural languages …"

What about determiners?

How to handle determiners, as in *A man loves Mary*?

Maybe interpret "a man" as $\exists$x.man(x) ?



S : (λx.love(x, mary))($\exists$x.man(x))
= love($\exists$x.man(x), mary)

VP : (λy.λx.love(x, y)(mary)
= λx.love(x, mary)

$\exists$x.man(x) ? — TV : λy.λx.love(x, y) — NP : mary

A man — *loves* — *Mary*

How do we *know* this is wrong?

$\exists$x.man(x) just doesn't mean "a man".

If anything it means "there is a man".

Analyzing determiners

Our goal is:

A man loves Mary $\rightarrow$ $\exists z\,(\text{man}(z) \wedge \text{love}(z, \text{mary}))$

$\exists z\,((\lambda y.\text{man}(y))(z) \wedge (\lambda x.\text{love}(x, \text{mary}))(z))$

What if we allow abstractions over any term?

$(\lambda Q.\exists z\,((\lambda y.\text{man}(y))(z) \wedge Q(z)))\,(\lambda x.\text{love}(x, \text{mary}))$

$(\lambda P.\lambda Q.\exists z\,(P(z) \wedge Q(z)))\,(\lambda x.\text{love}(x, \text{mary}))\,(\lambda y.\text{man}(y))$

Add to lexicon:

a $\rightarrow$ DT : $\lambda P.\lambda Q.\exists z\,(P(z) \wedge Q(z))$

And similarly:

every $\rightarrow$ DT : $\lambda P.\lambda Q.\forall z\,(P(z) \rightarrow Q(z))$

no $\rightarrow$ DT : $\lambda P.\lambda Q.\forall z\,(P(z) \rightarrow \neg Q(z))$

Determiners in action

S : $(\lambda Q.\exists z\,(man(z) \wedge Q(z)))(\lambda x.loves(x, mary))$
$= \exists z\,(man(z) \wedge (\lambda x.loves(x, mary))(z))$
$= \exists z\,(man(z) \wedge loves(z, mary))$

NP : $(\lambda P.\lambda Q.\exists z\,(P(z) \wedge Q(z)))(\lambda y.man(y))$
$= \lambda Q.\exists z\,((\lambda y.man(y))(z) \wedge Q(z))$
$= \lambda Q.\exists z\,(man(z) \wedge Q(z))$

VP : $(\lambda y.\lambda x.love(x, y)(mary)$
$= \lambda x.love(x, mary)$

DT : $\lambda P.\lambda Q.\exists z\,(P(z) \wedge Q(z))$

N : $\lambda y.man(y)$

TV : $\lambda y.\lambda x.love(x, y)$

NP : mary

A   *man*   *loves*   *Mary*

Add to lexicon:

$a \leftarrow$ DT : $\lambda P.\lambda Q.\exists z\,(P(z) \wedge Q(z))$
$man \leftarrow$ N : $\lambda y.man(y)$

Add to grammar:

NP : f(a) $\leftarrow$ DT : f   N : a
S : f(a) $\leftarrow$ NP : f   VP : a

different!

Type raising!

Wait, now how are we going to handle *John loves Mary*?

(λx.love(x, mary)) @ (john)	not systematic!
(john) @ (λx.love(x, mary))	not reducible!
(λP.P(john)) @ (λx.love(x, mary))	better?
= (λx.love(x, mary))(john)	
= love(john, mary)	yes!

So revise lexicon:

John $\leftarrow$ NP : λP.P(john)
Mary $\leftarrow$ NP : λP.P(mary)

This is called *type-raising:*

old type: e new type: (e$\rightarrow$t)$\rightarrow$t

The argument becomes the function!
(cf. callbacks, inversion of control)

Transitive verbs

We had this in our lexicon: *loves* $\leftarrow$ TV : $\lambda y.\lambda x.\text{love}(x, y)$

But if we now have: *Mary* $\leftarrow$ NP : $\lambda P.P(\text{mary})$

then *loves Mary* will be $(\lambda y.\lambda x.\text{love}(x, y))(\lambda P.P(\text{mary}))$

$= \lambda x.\text{love}(x, \lambda P.P(\text{mary}))$

Uh-oh! Solution?

Type-raising again! *loves* $\leftarrow$ TV : $\lambda R.\lambda x.R(\lambda y.\text{love}(x, y))$

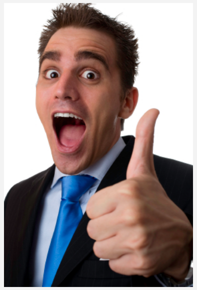
Old type for *loves*: $e \rightarrow (e \rightarrow t)$

New type for *loves*: $((e \rightarrow t) \rightarrow t) \rightarrow (e \rightarrow t)$

Let's see it in action …

Transitive verbs in action

S : (λP.P(john))(λx.love(x, mary))
= (λx.love(x, mary))(john)
= loves(john, mary)



VP : (λR.λx.R(λy.love(x, y)))(λQ.Q(mary))
= λx.(λQ.Q(mary))(λy.love(x, y))
= λx.(λy.love(x, y))(mary)
= λx.love(x, mary)

NP : λP.P(john)

TV : λR.λx.R(λy.love(x, y))

NP : λQ.Q(mary)

John *loves* *Mary*

Summing up

Our semantic lexicon covers many common syntactic types:

common nouns	*man*	$\leftarrow$	λx.man(x)
proper nouns	*Mary*	$\leftarrow$	λP.P(mary)
transitive verbs	*loves*	$\leftarrow$	λR.λx.R(λy.love(x, y))
intransitive verbs	*walks*	$\leftarrow$	λx.walk(x)
determiners	*a*	$\leftarrow$	λP.λQ.$\exists$z(P(z) $\wedge$ Q(z))

We can handle multiple phenomena in a uniform way!

Key ideas:

- extra λs for NPs
- abstraction over (i.e., introducing variables for) predicates
- inversion of control: subject NP as function, predicate VP as arg

Coordination

How to handle coordination, as in *John and Mary walk*?

What we'd *like* to get:

$$\text{walk(john)} \wedge \text{walk(mary)}$$

Already in our lexicon:

$$\begin{array}{lcl} \textit{John} & \leftarrow & \text{NP} : \lambda\text{P.P(john)} \\ \textit{Mary} & \leftarrow & \text{NP} : \lambda\text{Q.Q(mary)} \\ \textit{walk} & \leftarrow & \text{IV} : \lambda\text{x.walk(x)} \end{array}$$

Add to lexicon:

$$\textit{and} \quad \leftarrow \quad \text{CC} : \lambda\text{X.}\lambda\text{Y.}\lambda\text{R.(X(R)} \wedge \text{Y(R))}$$

My claim: this will work out just fine. Do you believe me?

Coordination in action

$(\lambda R.(R(john) \wedge R(mary))(\lambda x.walk(x))$
$= (\lambda x.walk(x))(john) \wedge (\lambda x.walk(x))(mary)$
$= walk(john) \wedge (\lambda x.walk(x))(mary)$
$= walk(john) \wedge walk(mary)$

$(\lambda Y.\lambda R.(R(john) \wedge Y(R)))(\lambda Q.Q(mary))$
$= \lambda R.(R(john) \wedge (\lambda Q.Q(mary))(R))$
$= \lambda R.(R(john) \wedge R(mary))$

$(\lambda X.\lambda Y.\lambda R.(X(R) \wedge Y(R)))(\lambda P.P(john))$
$= \lambda Y.\lambda R.((\lambda P.P(john))(R) \wedge Y(R))$
$= \lambda Y.\lambda R.(R(john) \wedge Y(R))$

$\lambda P.P(john)$ | $\lambda X.\lambda Y.\lambda R.(X(R) \wedge Y(R))$ | $\lambda Q.Q(mary)$ | $\lambda x.walk(x)$

John | *and* | *Mary* | *walk*

Other kinds of coordination

So great! We can handle coordination of NPs!

But what about coordination of …

intransitive verbs	*drinks and smokes*
transitive verbs	*washed and folded the laundry*
prepositions	*before and after the game*
determiners	*more than ten and less than twenty*

One solution is to have multiple lexicon entries for *and*

We'll let you work out the details …

Quantifier scope ambiguity

In this country, a woman gives birth every 15 minutes.
Our job is to find that woman and stop her.

— Groucho Marx celebrates quantifier scope ambiguity

$\exists w\,(\text{woman}(w) \land \forall f\,(\text{fifteen-minutes}(f) \rightarrow \text{gives-birth-during}(w, f)))$

$\forall f\,(\text{fifteen-minutes}(f) \rightarrow \exists w\,(\text{woman}(w) \land \text{gives-birth-during}(w, f)))$

Surprisingly, both readings are available in English!

Which one is the joke meaning?

Where scope ambiguity matters

Six sculptures — C, D, E, F, G, H — are to be exhibited in rooms 1, 2, and 3 of an art gallery.

- Sculptures C and E may not be exhibited in the same room.
- Sculptures D and G must be exhibited in the same room.
- If sculptures E and F are exhibited in the same room, no other sculpture may be exhibited in that room.
- At least one sculpture must be exhibited in each room, and no more than three sculptures may be exhibited in any room.

If sculpture D is exhibited in room 3 and sculptures E and F are exhibited in room 1, which of the following may be true?

A. Sculpture C is exhibited in room 1.
B. Sculpture H is exhibited in room 1.
C. Sculpture G is exhibited in room 2.
D. Sculptures C and H are exhibited in the same room.
E. Sculptures G and F are exhibited in the same room.

Scope need to be resolved!

At least one sculpture must be exhibited in each room.

The same sculpture in each room?

No more than three sculptures may be exhibited in any room.

Reading 1: For every room, there are no more than three sculptures exhibited in it.

Reading 2: At most three sculptures may be exhibited at all, regardless of which room.

Reading 3: The sculptures which can be exhibited in any room number at most three.
(For the other sculptures, there are restrictions on allowable rooms).

- Some readings will be ruled out by being uninformative or by contradicting other statements
- Otherwise we must be content with distributions over scope-resolved semantic forms

Classic example

Every man loves a woman

Reading 1: the women may be different

$\forall x\ (man(x) \rightarrow \exists y\ (woman(y) \wedge love(x, y)))$

Reading 2: there is one particular woman

$\exists y\ (woman(y) \wedge \forall x\ (man(x) \rightarrow love(x, y)))$

What does our system do?

Scope ambiguity in action

$(\lambda Q.\forall z\,(man(z) \rightarrow Q(z)))(\lambda x.\exists w\,(woman(w) \wedge love(x, w)))$
$= \forall z\,(man(z) \rightarrow (\lambda x.\exists w\,(woman(w) \wedge love(x, w)))(z))$
$= \forall z\,(man(z) \rightarrow \exists w\,(woman(w) \wedge love(z, w)))$

$(\lambda R.\lambda x.R(\lambda y.love(x, y)))(\lambda Q.\exists w\,(woman(w) \wedge Q(w)))$
$= \lambda x.(\lambda Q.\exists w\,(woman(w) \wedge Q(w)))(\lambda y.love(x, y))$
$= \lambda x.\exists w\,(woman(w) \wedge (\lambda y.love(x, y))(w))$
$= \lambda x.\exists w\,(woman(w) \wedge love(x, w))$

$(\lambda P.\lambda Q.\forall z\,(P(z) \rightarrow Q(z)))(\lambda y.man(y))$
$= \lambda Q.\forall z\,((\lambda y.man(y))(z) \rightarrow Q(z))$
$= \lambda Q.\forall z\,(man(z) \rightarrow Q(z))$

$(\lambda P.\lambda Q.\exists w\,(P(w) \wedge Q(w)))(\lambda x.woman(x))$
$= \lambda Q.\exists w\,((\lambda x.woman(x))(w) \wedge Q(w))$
$= \lambda Q.\exists w\,(woman(w) \wedge Q(w))$

$\lambda P.\lambda Q.\forall z\,(P(z) \rightarrow Q(z))$ $\lambda y.man(y)$ $\lambda R.\lambda x.R(\lambda y.love(x, y))$ $\lambda P.\lambda Q.\exists w\,(P(w) \wedge Q(w))$ $\lambda x.woman(x)$

Every *man* *loves* *some* *woman*

nltk.sem [Garrette & Klein 2008]

The nltk.sem package contains Python code for:

- First-order logic & typed lambda calculus
- Theorem proving, model building, & model checking
- DRT & DRSs
- Cooper storage, hole semantics, glue semantics
- Linear logic
- A (partial) implementation of Chat-80!

http://nltk.googlecode.com/svn/trunk/doc/api/nltk.sem-module.html

nltk.sem.logic

```
>>> import nltk
>>> from nltk.sem import logic
>>> logic.demo()
>>> parser = logic.LogicParser(type_check=True)

>>> man = parser.parse("\ y.man(y)")
>>> woman = parser.parse("\ x.woman(x)")
>>> love = parser.parse("\ R x.R(\ y.love(x,y))")
>>> every = parser.parse("\ P Q.all x.(P(x) -> Q(x))")
>>> some = parser.parse("\ P Q.exists x.(P(x) & Q(x))")

>>> every(man).simplify()
<LambdaExpression \Q.all x.(man(x) -> Q(x))>

>>> love(some(woman)).simplify()
<LambdaExpression \x.exists z.(woman(z) & love(x, z))>

>>> every(man)(love(some(woman))).simplify()
<AllExpression all x.(man(x) -> exists z.(woman(z) & love(x, z)))>
```

What's missing?

OK, this all seems super duper, but … what's missing?

Can we solve these NLU challenges yet?

Why not?

Six sculptures — C, D, E, F, G, H — are to be exhibited in rooms 1, 2, and 3 of an art gallery.

- Sculptures C and E may not be exhibited in the same room.
- Sculptures D and G must be exhibited in the same room.
- If sculptures E and F are exhibited in the same room, no other sculpture may be exhibited in that room.
- At least one sculpture must be exhibited in each room, and no more than three sculptures may be exhibited in any room.

If sculpture D is exhibited in room 3 and sculptures E and F are exhibited in room 1, which of the following may be true?

A. Sculpture C is exhibited in room 1.
B. Sculpture H is exhibited in room 1.
C. Sculpture G is exhibited in room 2.
D. Sculptures C and H are exhibited in the same room.
E. Sculptures G and F are exhibited in the same room.

Yes, hi, I need to book a flight for myself and my husband from Boston to SFO, or Oakland would be OK too. We need to fly out on Friday the 12th, and then I could fly back on Sunday evening or Monday morning, but he won't return until Wednesday the 18th, because he's staying for business. No flights with more than one stop, and we don't want to fly on United because we hate their guts.