Introduction to semantic parsing

Bill MacCartney
CS224U
27 April 2015
Reminder

Lit Review due in nine days! (May 6)

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, ...
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 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 corn machines!

```
$ ssh corn.stanford.edu
$ cd /afs/ir/class/cs224n/src/chat/
$ module load sicstus
$ sicstus
? [load].
? hi.
? what is the capital of france?
```

Sample queries can be found at:\n
```
/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

Brittle, narrow coverage

Robust, broad coverage

Fuzzy, partial understanding

SHRDLU  CHAT-80
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 = 2014  
ORDER BY co2_emissions.volume DESC  
LIMIT 1;
```
For a robot control application, you might want a custom-designed procedural language:

\[
\text{Go to the third junction and take a left.}
\]

\[
\text{(do-sequentially}
\]
\[
\text{ (do-n-times 3}
\]
\[
\text{ (do-sequentially}
\]
\[
\text{ (move-to forward-loc)
\]
\[
\text{ (do-until}
\]
\[
\text{ (junction current-loc)
\]
\[
\text{ (move-to forward-loc)))}
\]
\[
\text{(turn-left))}
\]
## Intents and arguments

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

<table>
<thead>
<tr>
<th>Intent Description</th>
<th>Intents and Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>directions to SF by train</td>
<td>(TravelQuery (Destination /m/0d6lp) (Mode TRANSIT))</td>
</tr>
<tr>
<td>angelina jolie net worth</td>
<td>(FactoidQuery (Entity /m/0f4vbz) (Attribute /person/net_worth))</td>
</tr>
<tr>
<td>text my wife on my way</td>
<td>(SendMessage (Recipient 0x31cbf492) (MessageType SMS) (Subject &quot;on my way&quot;))</td>
</tr>
<tr>
<td>play sunny by boney m</td>
<td>(PlayMedia (MediaType MUSIC) (SongTitle &quot;sunny&quot;) (MusicArtist /m/017mh))</td>
</tr>
<tr>
<td>weather friday austin tx</td>
<td>(WeatherQuery (Location /m/0vzm) (Date 2013-12-13))</td>
</tr>
<tr>
<td>is REI open on sunday</td>
<td>(LocalQuery (QueryType OPENING_HOURS) (Location /m/02nx4d) (Date 2013-12-15))</td>
</tr>
</tbody>
</table>
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.

\[
\begin{align*}
\text{John walks} & \quad \text{walk}(\text{john}) \\
\text{John loves Mary} & \quad \text{love}(\text{john}, \text{mary}) \\
\text{Every man loves Mary} & \quad \forall x \ (\text{man}(x) \rightarrow \text{love}(x, \text{mary}))
\end{align*}
\]

(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: $\land$, $\lor$, $\neg$, $\rightarrow$
  - Quantifiers: $\forall$, $\exists$
  - Other punctuation: parens, commas

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

“content words” (user-defined)

“function words”
An NLU pipeline

- English sentences
  
  *John smokes. Everyone who smokes snores.*

- Syntactic analysis
  
  \[(S \ (NP \ John) \ (VP \ smokes))\]

- Semantic analysis
  
  \[\text{smoke}(john)\]

- Inference / execution
  
  \[\forall x. \text{smoke}(x) \rightarrow \text{snore}(x), \text{smoke}(john)\]
  \[\Rightarrow \text{snore}(john)\]

Focus of semantic parsing
From language to logic

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

<table>
<thead>
<tr>
<th>Natural Language</th>
<th>Logical Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>John walks</td>
<td>walk(john)</td>
</tr>
<tr>
<td>John loves Mary</td>
<td>love(john, mary)</td>
</tr>
<tr>
<td>A man walks</td>
<td>$\exists x. \text{man}(x) \land \text{walk}(x)$</td>
</tr>
<tr>
<td>A man loves Mary</td>
<td>$\exists x. \text{man}(x) \land \text{love}(x, \text{mary})$</td>
</tr>
<tr>
<td>John and Mary walk</td>
<td>walk(john) $\land$ walk(mary)</td>
</tr>
<tr>
<td>Every man walks</td>
<td>$\forall x. \text{man}(x) \rightarrow \text{walk}(x)$</td>
</tr>
<tr>
<td>Every man loves a woman</td>
<td>$\forall x. \text{man}(x) \rightarrow \exists y. \text{woman}(y) \land \text{love}(x, y)$</td>
</tr>
</tbody>
</table>

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?

<table>
<thead>
<tr>
<th>English</th>
<th>French</th>
</tr>
</thead>
<tbody>
<tr>
<td>John walks</td>
<td>Jean marche</td>
</tr>
<tr>
<td>John loves Mary</td>
<td>Jean aime Marie</td>
</tr>
<tr>
<td>A man walks</td>
<td>Un homme marche</td>
</tr>
<tr>
<td>A man loves Mary</td>
<td>Un homme aime Marie</td>
</tr>
<tr>
<td>John and Mary walk</td>
<td>Jean et Marie marche</td>
</tr>
<tr>
<td>Every man walks</td>
<td>Chaque homme marche</td>
</tr>
<tr>
<td>Every man loves a woman</td>
<td>Chaque homme aime une femme</td>
</tr>
</tbody>
</table>

In MT, we break the input into pieces, translate the pieces, and then put the pieces back together.
## A logical lexicon (first attempt)

<table>
<thead>
<tr>
<th>English</th>
<th>Symbolic Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>John walks</td>
<td>( \text{walk}(\text{john}) )</td>
</tr>
<tr>
<td>John loves Mary</td>
<td>( \text{love}(\text{john}, \text{mary}) )</td>
</tr>
<tr>
<td>A man walks</td>
<td>( \exists x. \text{man}(x) \land \text{walk}(x) )</td>
</tr>
<tr>
<td>A man loves Mary</td>
<td>( \exists x. \text{man}(x) \land \text{love}(x, \text{mary}) )</td>
</tr>
<tr>
<td>John and Mary walk</td>
<td>( \text{walk}(\text{john}) \land \text{walk}(\text{mary}) )</td>
</tr>
<tr>
<td>Every man walks</td>
<td>( \forall x. \text{man}(x) \rightarrow \text{walk}(x) )</td>
</tr>
<tr>
<td>Every man loves a woman</td>
<td>( \forall x. \text{man}(x) \rightarrow \exists y. \text{woman}(y) \land \text{love}(x, y) )</td>
</tr>
</tbody>
</table>

**Symbols**

- \( \text{John} : \text{john} \)
- \( \text{Mary} : \text{mary} \)
- \( \text{walks} : \text{walk}(?\)\)
- \( \text{loves} : \text{love}(?, ?) \)
- \( \text{man} : \text{man}(?) \)
- \( \text{woman} : \text{woman}(?) \)
- \( \text{and} : \land \)
- \( a : \exists x. ? \land ? \)
- \( \text{every} : \forall x. ? \rightarrow ? \)
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

The meaning of the whole is determined by the meanings of the parts and the manner in which they are combined.
Example: syntactic analysis

```
S
├── NP
│   └── John
├── VP
│   ├── TV
│   │   └── loves
│   └── NP
│       └── Mary
└── NP
```
Example: semantic lexicon

S
  /   \\    \    \\   \\   \\    \\     \\
NP : john  TV : love(?, ?)  NP : mary
  /   \\    \\    \\    \\    \\    \\      \\
 John  loves  Mary
Example: semantic composition

$$S \rightarrow \text{NP : john} \quad \text{VP : love(?, mary)} \quad \text{NP : mary}$$

- John
- loves
- Mary
Example: semantic composition

S : love(john, mary)

NP : john
  |  John
  |
TV : love(?, ?)
  |  loves
  |
NP : mary
  |  Mary

VP : love(?, mary)
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 represent 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 $\lambda$ to bind free variables
  $\lambda x.\text{love}(x, \text{mary})$  
  \textit{loves Mary}
- The new meta-logical symbol $\lambda$ marks missing information in the object language ($\lambda$-)FOL
- We \textit{abstract} over $x$
- Just like in programming languages!
  
  \textbf{Python:} \hspace{1cm} \texttt{lambda x: x \% 2 == 0}
  \textbf{Ruby:} \hspace{1cm} \texttt{lambda \{|x| x \% 2 == 0\}}
- How do we combine these new formulae and terms?
Super glue

- We’ll combine semantic fragments via function application
  
  \[(\lambda x.\text{love}(x, \text{mary})) \ @ \ \text{john}\]
  
  \[(\lambda x.\text{love}(x, \text{mary}))(\text{john})\]

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

- Function application triggers \textit{beta reduction}
  replace the \(\lambda\)-bound variable by the argument throughout the body
Beta reduction

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

1. Strip off the \(\lambda\) prefix

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

2. Remove the argument

\(\text{love}(x, \text{mary})\)

3. Replace all occurrences of \(\lambda\)-bound variable by argument

\(\text{love}(\text{john}, \text{mary})\)
Semantic construction with lambdas

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

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

\[ \text{NP : john} \]
\[ \downarrow \]
\[ \text{John} \]

\[ \text{NP : mary} \]
\[ \downarrow \]
\[ \text{Mary} \]

\[ \text{TV : } \lambda y.\lambda x.\text{love}(x, y) \]
\[ \downarrow \]
\[ \text{loves} \]
A semantic grammar

Lexicon

John ← NP : john
Mary ← NP : mary
loves ← TV : \( \lambda y. \lambda x. \text{love}(x, y) \)

Composition rules

VP : \( f(a) \) → TV : \( f \)   NP : \( a \)
S : \( f(a) \) → NP : \( a \)   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 ...”
Appendix

(The rest of this deck contains more detail on using lambda calculus for semantic construction, and should be considered optional.)
Application vs. abstraction

$(\lambda x.\text{love}(x, \text{mary})) (\text{john})$
What about determiners?

How to handle determiners, as in *A man loves Mary*?
Maybe interpret “a man” as \( \exists x. \text{man}(x) \) ?

\[
\begin{align*}
S & : (\lambda x. \text{love}(x, \text{mary}))(\exists x. \text{man}(x)) \\
& = \text{love}(\exists x. \text{man}(x), \text{mary})
\end{align*}
\]

\[
\begin{align*}
\exists x. \text{man}(x) & ? \\
\text{VP} & : (\lambda y.\lambda x. \text{love}(x, y))(\text{mary}) \\
& = \lambda x. \text{love}(x, \text{mary})
\end{align*}
\]

How do we *know* this is wrong?

\( \exists x. \text{man}(x) \) just doesn’t mean “a man”. If anything it means “there is a man”.

\[
\begin{align*}
\exists x. \text{man}(x) & ? \\
\text{TV} & : \lambda y.\lambda x. \text{love}(x, y) \\
\text{NP} & : \text{mary}
\end{align*}
\]

\[
\begin{align*}
A \text{ man} & \\
\text{loves} & \\
\text{Mary} &
\end{align*}
\]
Analyzing determiners

Our goal is:

\[
A \text{ man loves } \text{ Mary} \rightarrow \exists z \ (\text{man}(z) \land \text{love}(z, \text{mary})) \\
\exists z \ ((\lambda y.\text{man}(y))(z) \land (\lambda x.\text{love}(x, \text{mary}))(z))
\]

What if we allow abstractions over any term?

\[
(\lambda Q. \exists z \ ((\lambda y.\text{man}(y))(z) \land Q(z)) \ (\lambda x.\text{love}(x, \text{mary}))) \\
(\lambda P.\lambda Q. \exists z \ (P(z) \land Q(z))) (\lambda x.\text{love}(x, \text{mary})) (\lambda y.\text{man}(y))
\]

Add to lexicon:

\[
a \rightarrow \text{DT} : \lambda P.\lambda Q. \exists z \ (P(z) \land Q(z))
\]

And similarly:

\[
\text{every} \rightarrow \text{DT} : \lambda P.\lambda Q. \forall z \ (P(z) \rightarrow Q(z)) \\
\text{no} \rightarrow \text{DT} : \lambda P.\lambda Q. \forall z \ (P(z) \rightarrow \neg Q(z))
\]
Determiners in action

\[
S : (\lambda Q. \exists z (\text{man}(z) \land Q(z))) (\lambda x. \text{loves}(x, \text{mary}))
= \exists z (\text{man}(z) \land (\lambda x. \text{loves}(x, \text{mary}))(z))
= \exists z (\text{man}(z) \land \text{loves}(z, \text{mary}))
\]

Add to lexicon:

\[
a \leftarrow \text{DT} : \lambda P. \lambda Q. \exists z (P(z) \land Q(z))
\]

\[
\text{man} \leftarrow \text{N} : \lambda y. \text{man}(y)
\]

Add to grammar:

\[
\text{NP} : f(a) \leftarrow \text{DT} : f \quad \text{N} : a
\]

\[
\text{S} : f(a) \leftarrow \text{NP} : f \quad \text{VP} : a
\]

\[
\text{different!}
\]
Type raising!

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

\[ (\lambda x.\text{love}(x, \text{mary})) \circ (\text{john}) \]

not systematic!

\[ (\text{john}) \circ (\lambda x.\text{love}(x, \text{mary})) \]

not reducible!

\[ (\lambda P.P(\text{john})) \circ (\lambda x.\text{love}(x, \text{mary})) \]

better?

\[ = (\lambda x.\text{love}(x, \text{mary}))(\text{john}) \]

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

yes!

So revise lexicon:

\[ \text{John} \leftarrow \text{NP} : \lambda P.P(\text{john}) \]

\[ \text{Mary} \leftarrow \text{NP} : \lambda P.P(\text{mary}) \]

This is called *type-raising*:

old type: e  new type: (e→t)→t

The argument becomes the function!

(cf. callbacks, inversion of control)
Transitive verbs

We had this in our lexicon:

\[ \text{loves} \leftarrow \text{TV} : \lambda y.\lambda x.\text{love}(x, y) \]

But if we now have:

\[ \text{Mary} \leftarrow \text{NP} : \lambda P.P(\text{mary}) \]

then \( \text{loves Mary} \) 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!

\[ \text{loves} \leftarrow \text{TV} : \lambda R.\lambda x.R(\lambda y.\text{love}(x, y)) \]

Old type for \( \text{loves} \):

\[ e \rightarrow (e \rightarrow t) \]

New type for \( \text{loves} \):

\[ ((e \rightarrow t) \rightarrow t) \rightarrow (e \rightarrow t) \]

Let’s see it in action …
Transitive verbs in action

\[ S : (\lambda.P.P(john))(\lambda.x.\text{love}(x, \text{mary})) = (\lambda.x.\text{love}(x, \text{mary}))(\text{john}) = \text{loves}(\text{john, mary}) \]

\[ \text{NP : } \lambda.P.P(john) \]
- \text{John}

\[ \text{TV : } \lambda.R.\lambda.x.R(\lambda.y.\text{love}(x, y))(\lambda.Q.Q(mary)) = \lambda.x.(\lambda.Q.Q(mary))(\lambda.y.\text{love}(x, y)) = \lambda.x.(\lambda.y.\text{love}(x, y))(\text{mary}) = \lambda.x.\text{love}(x, \text{mary}) \]
- \text{loves}

\[ \text{NP : } \lambda.Q.Q(mary) \]
- \text{Mary}

\[ \text{VP : } (\lambda.R.\lambda.x.R(\lambda.y.\text{love}(x, y))(\lambda.Q.Q(mary))) \]

\[ \text{S : } (\lambda.P.P(john))(\lambda.x.\text{love}(x, \text{mary}))(\text{john}) = \text{loves}(\text{john, mary}) \]
Summing up

Our semantic lexicon covers many common syntactic types:

- common nouns: \( \text{man} \leftarrow \lambda x. \text{man}(x) \)
- proper nouns: \( \text{Mary} \leftarrow \lambda P. P(\text{mary}) \)
- transitive verbs: \( \text{loves} \leftarrow \lambda R. \lambda x. R(\lambda y. \text{love}(x, y)) \)
- intransitive verbs: \( \text{walks} \leftarrow \lambda x. \text{walk}(x) \)
- determiners: \( \text{a} \leftarrow \lambda P. \lambda Q. \exists z (P(z) \land Q(z)) \)

We can handle multiple phenomena in a uniform way!

Key ideas:
- extra \( \lambda \)s 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:

\[
\text{walk(john)} \land \text{walk(mary)}
\]

Already in our lexicon:

\[
\begin{align*}
\text{John} & \leftarrow \text{NP} : \lambda P. P(\text{john}) \\
\text{Mary} & \leftarrow \text{NP} : \lambda Q. Q(\text{mary}) \\
\text{walk} & \leftarrow \text{IV} : \lambda x. \text{walk}(x)
\end{align*}
\]

Add to lexicon:

\[
\text{and} \leftarrow \text{CC} : \lambda X. \lambda Y. \lambda R. (X(R) \land Y(R))
\]

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

\[
\lambda. (R(\text{john}) \land R(\text{mary}))(\lambda x. \text{walk}(x))
\]

= \((\lambda x. \text{walk}(x))(\text{john}) \land (\lambda x. \text{walk}(x))(\text{mary})\)

= \text{walk}(\text{john}) \land (\lambda x. \text{walk}(x))(\text{mary})

= \text{walk}(\text{john}) \land \text{walk}(\text{mary})
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
\[
\forall x (\text{man}(x) \rightarrow \exists y (\text{woman}(y) \land \text{love}(x, y)))
\]

Reading 2: there is one particular woman
\[
\exists y (\text{woman}(y) \land \forall x (\text{man}(x) \rightarrow \text{love}(x, y)))
\]

What does our system do?
Scope ambiguity in action

\[(\lambda Q. \forall z (\text{man}(z) \rightarrow Q(z))) (\lambda x. \exists w (\text{woman}(w) \land \text{love}(x, w)))\]
\[= \forall z (\text{man}(z) \rightarrow (\lambda x. \exists w (\text{woman}(w) \land \text{love}(x, w)))(z))\]
\[= \forall z (\text{man}(z) \rightarrow \exists w (\text{woman}(w) \land \text{love}(z, w)))\]

\[= (\lambda R. \lambda x. R(\lambda y. \text{love}(x, y)))(\lambda Q. \exists w (\text{woman}(w) \land Q(w)))\]
\[= \lambda x. (\lambda Q. \exists w (\text{woman}(w) \land Q(w)))(\lambda y. \text{love}(x, y))\]
\[= \lambda x. \exists w (\text{woman}(w) \land (\lambda y. \text{love}(x, y))(w))\]
\[= \lambda x. \exists w (\text{woman}(w) \land \text{love}(x, w))\]

\[= (\lambda P. \lambda Q. \forall z (P(z) \rightarrow Q(z))) (\lambda y. \text{man}(y))\]
\[= \lambda Q. \forall z ((\lambda y. \text{man}(y))(z) \rightarrow Q(z))\]
\[= \lambda Q. \forall z (\text{man}(z) \rightarrow Q(z))\]

\[= (\lambda P. \lambda Q. \exists w (P(w) \land Q(w))) (\lambda x. \text{woman}(x))\]
\[= \lambda Q. \exists w ((\lambda x. \text{woman}(x))(w) \land Q(w))\]
\[= \lambda Q. \exists w (\text{woman}(w) \land Q(w))\]

\[\lambda P. \lambda Q. \forall z (P(z) \rightarrow Q(z)) \quad \lambda y. \text{man}(y) \quad \lambda R. \lambda x. R(\lambda y. \text{love}(x, y)) \quad \lambda P. \lambda Q. \exists w (P(w) \land Q(w)) \lambda x. \text{woman}(x)\]

\[\text{Every} \quad \text{man} \quad \text{loves} \quad \text{some} \quad \text{woman}\]
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!

>>> 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.