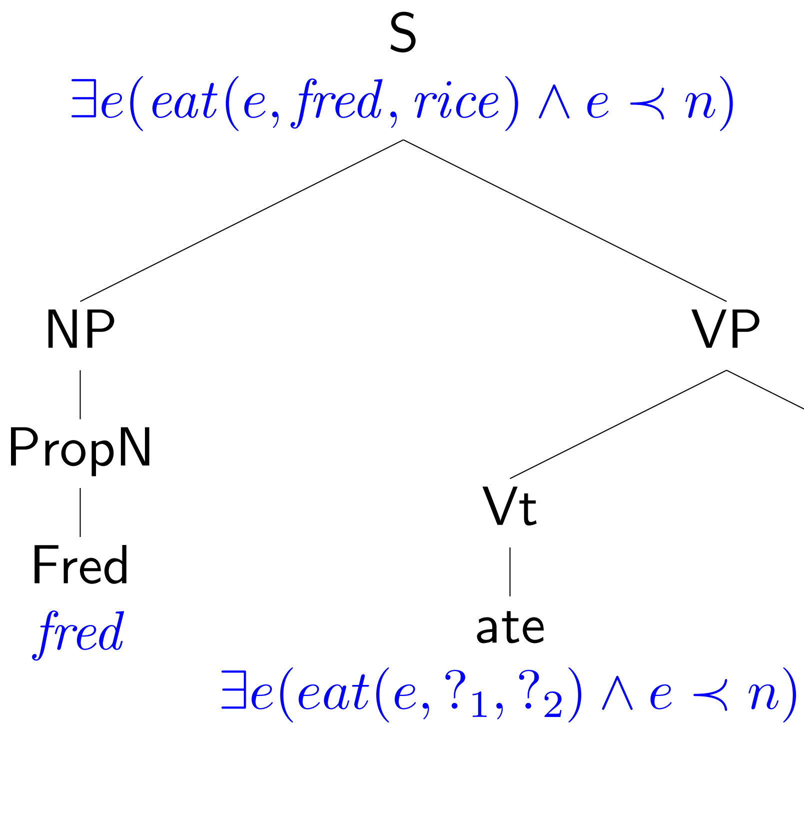
Foundations of Natural Language Processing
Lecture 18c
Compositional Semantics: Technical Details

Alex Lascarides

School of informatics

Last Time: Deriving FoL LFs via Linguistic Syntax



- How do we get the bits to combine?
  What are the LFs of the intermediate nodes?

Today: How it's done: Lambda Calculus.

Lambda Calculus and Beta Reduction

Allows us to work with 'partially constructed' formulae!

- If φ is a well-formed FoL expression and x is a variable, then $\lambda x \varphi$ is a well-formed FoL expression. It's a function, known as a λ-term.

- λ-terms have interesting semantics, but they also offer a way of substituting (free) variables in an FoL expression with values.

$$\lambda x \varphi(a) = \varphi[x/a]$$

- Creating a function $\lambda x \varphi$ from an expresion φ is called Lambda (λ) abstraction
Function application is called Beta (β) reduction.

Example:

- $\lambda y \lambda x(\exists e(eat(e, x, y) \wedge e \prec n))(rice)$ becomes
$\lambda x(\exists e(eat(e, x, rice) \wedge e \prec n))$

Introducing variables corresponding to properties, relations. . .

- If we introduces variables of 'higher type' then we can substitute variables corresponding to properties, relations etc with values that can be λ-terms!!

- $\lambda P.P(fred)$:
  the properties of Fred (man, tall,. . .)
  $\lambda P.P(fred)(man)$ becomes $man(fred)$

An example where the argument is a λ-term:

- $\lambda P(P(fred))(\lambda x(\exists e(eat(e,x,rice) \wedge e \prec n)))$ becomes
  $\lambda x(\exists e(eat(e,x,rice) \wedge e \prec n))(fred)$ becomes
  $\exists e(eat(e,fred,rice) \wedge e \prec n)$

Example Composition for Fred ate rice

But we'll see in a minute why it's problematic. . .
. . . and why λ-abstraction on higher types provides a solution!

- S: $\lambda x \exists e(eat(e, x, rice) \wedge e \prec n)(fred)$, $\exists e(eat(e, fred, rice) \wedge e \prec n)$
  - NP: $fred$
    - PropN: $fred$
      - Fred: $fred$
  - VP: $\lambda y \lambda x \exists e(eat(e, x, y) \wedge e \prec n)(rice)$, $\lambda x \exists e(eat(e, x, rice) \wedge e \prec n)$
    - Vt: $\lambda y \lambda x \exists e(eat(e, x, y) \wedge e \prec n)$
      - ate: $\lambda y \lambda x \exists e(eat(e, x, y) \wedge e \prec n)$
    - NP: $rice$
      - MassN: $rice$
        - rice: $rice$

The Grammar that generates that tree

S $\rightarrow$ NP VP *VP.Sem(NP.Sem)*	**(Sentences)**
NP $\rightarrow$ MassN *MassN.Sem* | PropN *PropN.Sem*	**(Noun phrases)**
VP $\rightarrow$ Vi *Vi.Sem* | Vt NP *Vt.Sem(NP.Sem)*	**(Verb phrases)**
PropN $\rightarrow$ Fred *fred* | Jo *jo*. . .	**(Proper nouns)**
MassN $\rightarrow$ rice *rice* | wood *wood* . . .	**(Mass nouns)**
Vi $\rightarrow$ talked $\lambda x \exists e(talk(e, x) \wedge e \prec n)$ | . . .	**(Intransitive verbs)**
Vt $\rightarrow$ ate $\lambda y \lambda x.\exists e(eat(e, x, y) \wedge e \prec n)$ | . . .	**(Transitive verbs)**

Observations:

- λ-term for Vt ensures NP values are in right positions to predicate *eat*
- Rules with two daughters specify in semantics which daughter is the functor and which the argument
  - S rule: VP is the functor.
  - Transitive VP rule: Vt is the functor.
- Unary rules 'pass up' the semantics from the daughter.

Problematic!

Every man ate rice: $\forall x(man(x) \rightarrow \exists e(eat(e,x,rice) \wedge e \prec n))$

Breaking it down:

- What is the meaning of *Every man* anyway?
  $\forall x(man(x) \rightarrow Q(x))$

- If so, the subject NP needs to be:
  $\lambda Q \forall x(man(x) \rightarrow Q(x))$

- But in our grammar we had the VP as the functor:
  S $\rightarrow$ NP VP *VP.Sem(NP.Sem)*

- $\lambda z \exists e(eat(e,z,rice) \wedge e \prec n)(\lambda Q \forall x(man(x) \rightarrow Q(x)))$ becomes
  $\lambda z \exists e(eat(e, \lambda Q \forall x(man(x) \rightarrow Q(x)), rice) \wedge e \prec n)$

- That's not even syntactically well-formed!!

Solution

Make NP the functor and VP the argument.

S $\rightarrow$ NP VP *NP.Sem(VP.Sem)*

$$\lambda Q \forall x(man(x) \rightarrow Q(x))(\lambda z \exists e(eat(e, z, rice) \wedge e \prec n))$$
$$\forall x(man(x) \rightarrow \lambda z \exists e(eat(e, z, rice) \wedge e \prec n))(x))$$
$$\forall x(man(x) \rightarrow \exists e(eat(e, x, rice) \wedge e \prec n))$$

But this means NPs must all look like this: $\lambda P.P(x)$.
Fred $\mapsto \lambda P.P(fred)$ etc.

Now a problem with transitive verbs!!

ate	every grape:
$\lambda y \lambda z \exists e(eat(e,z,y) \wedge e \prec n)$	$\lambda Q \forall x(grape(x) \rightarrow Q(x))$

NP.Sem(Vt.Sem) is ill formed!

$\lambda Q \forall x(grape(x) \rightarrow Q(x))(\lambda y \lambda z \exists e(eat(e,z,y) \wedge \prec n))$ becomes
$\forall x(grape(x) \rightarrow \lambda y \lambda z \exists e(eat(e,z,y) \wedge e \prec n)(x)$ becomes
$\forall x(grape(x) \rightarrow \lambda z \exists e(eat(e,z,x) \wedge e \prec n))$ ill-formed!

It should be: $\lambda z \forall x(grape(x) \rightarrow \exists e(eat(e,z,x) \wedge e \prec n))$

Type Raising to the rescue again

VP $\rightarrow$ Vt NP *Vt.Sem(NP.Sem)*
Vt $\rightarrow$ ate $\lambda R.\lambda z.R(\lambda y.\exists e(eat(e,z,y) \wedge e \prec n))$

ate every grape:

$\lambda R.\lambda z.R(\lambda y.\exists e(eat(e,z,y) \wedge e \prec n))(\lambda Q \forall x(grape(x) \rightarrow Q(x)))$ becomes
$\lambda z \lambda Q \forall x(grape(x) \rightarrow Q(x))(\lambda y.\exists e(eat(e,z,y) \wedge e \prec n))$ becomes
$\lambda z \forall x(grape(x) \rightarrow \lambda y.\exists e(eat(e,z,y) \wedge e \prec n)(x))$ becomes
$\lambda z \forall x(grape(x) \rightarrow \exists e(eat(e,z,x)))$

Grammar Refined! (Changes in purple)

S $\rightarrow$	NP VP *NP.Sem(VP.Sem)*	**(Sentences)**
NP $\rightarrow$	MassN *MassN.Sem* | PropN *PropN.Sem* | Det N *Det.Sem(N.Sem)*	**(Noun phrases)**
VP $\rightarrow$	Vi *Vi.Sem* | Vt NP *Vt.Sem(NP.Sem)*	**(Verb phrases)**
PropN $\rightarrow$	Fred $\lambda P.P(fred)$ | ...	**(Proper nouns)**
MassN $\rightarrow$	rice $\lambda P.P(rice)$ | ...	**(Mass nouns)**
Vi $\rightarrow$	talked $\lambda x \exists e(talk(e, x) \wedge e \prec n)$ | ...	**(Intransitive verbs)**
Vt $\rightarrow$	ate $\lambda R.\lambda z.R(\lambda y.\exists e(eat(e, z, y) \wedge e \prec n))$	**(Transitive verbs)**
N $\rightarrow$	man $\lambda x.man(x)$	**(Count Nouns)**
Det $\rightarrow$	a $\lambda P \lambda Q \exists x(P(x) \wedge Q(x))$ | every $\lambda Q \lambda Q \exists x(P(x) \rightarrow Q(x))$	**(Determiners)**

Example Derivation: Every man ate rice

S

$\lambda Q.\forall x(man(x) \rightarrow Q(x))(\lambda z.\exists e(eat(e,z,rice) \wedge e \prec n))$

$\forall x(man(x) \rightarrow \lambda z.\exists e(eat(e,z,rice) \wedge e \prec n)(x))$

$\forall x(man(x) \rightarrow \exists e(eat(e,x,rice) \wedge e \prec n))$

- NP
  - $\lambda P \lambda Q \forall x(P(x) \rightarrow Q(x))(\lambda z.man(z))$
  - $\lambda Q \forall x(\lambda z.man(z)(x) \rightarrow Q(x))$
  - $\lambda Q \forall x(man(x) \rightarrow Q(x))$
  - Det — every — $\lambda P \lambda Q \forall x(P(x) \rightarrow Q(x))$
  - N — man — $\lambda z.man(z)$
- VP
  - $\lambda R.\lambda z.R(\lambda y \exists e(eat(e,z,y) \wedge e \prec n))(\lambda P.P(rice))$
  - $\lambda z.\lambda P.P(rice)(\lambda y \exists e(eat(e,z,y) \wedge e \prec n))$
  - $\lambda z.(\lambda y.\exists e(eat(e,z,y) \wedge e \prec n)(rice))$
  - $\lambda z.\exists e(eat(e,z,rice) \wedge e \prec n)$
  - Vt — ate — $\lambda R.\lambda z.R(\lambda y \exists e(eat(e,z,y) \wedge e \prec n))$
  - NP — MassN — rice — $\lambda P.P(rice)$

Every man loves a woman — Other reading??

- S
  $\lambda Q.\forall x(man(x) \rightarrow Q(x))(\lambda z.\exists w(woman(w) \wedge \exists e(love(e,z,w) \wedge n \subseteq e)))$
  $\forall x(man(x) \rightarrow \lambda z.\exists w(woman(w) \wedge \exists e(love(e,z,w) \wedge n \subseteq e)))$
  $\forall x(man(x) \rightarrow \exists w(woman(w) \wedge \exists e(love(e,x,w) \wedge n \subseteq e)))$
  - NP
    $\lambda P \lambda Q(P(x) \rightarrow Q(x))(\lambda z.man(z))$
    $\lambda Q(\lambda z.man(z)(x) \rightarrow Q(x))$
    $\lambda Q(man(x) \rightarrow Q(x))$
    - Det — every
      $\lambda P \lambda Q \forall x(P(x) \rightarrow Q(x))$
    - N — man
      $\lambda z.man(z)$
  - VP
    $\lambda R.\lambda z.R(\lambda y \exists e(love(e,z,y) \wedge n \subseteq e))(\lambda T \exists w(woman(w) \wedge T(w))) \mapsto$
    $\lambda z.\exists w(woman(w) \wedge \exists e(love(e,z,w) \wedge n \subseteq e))$
    - Vt — loves
      $\lambda R.\lambda z.R(\lambda y \exists e(love(e,z,y) \wedge n \subseteq e))$
    - NP
      $\lambda S \lambda T \exists w(S(w) \wedge T(w))(\lambda z.woman(z))$
      $\lambda T \exists w(woman(w) \wedge T(w))$
      - DET — a
        $\lambda S.\lambda T \exists w(S(w) \wedge T(w))$
      - N — woman
        $\lambda z.woman(z)$

Semantic Ambiguity

- Every man loves a woman has two different interpretations because of its determiners:
  - Possibly a different woman per man
    $\forall x(man(x) \rightarrow \exists y(woman(y) \wedge \exists e(love(e, x, y) \wedge n \subseteq e)))$
  - The same woman for all men
    $\exists y(woman(y) \wedge \forall x(man(x) \rightarrow \exists e(love(e, x, y) \wedge n \subseteq e)))$

- But the English sentence isn't syntactically ambiguous!!

Scope

- The ambiguity arises because every and a each has its own **scope**:

  Interpretation 1: every **has scope over** a
  Interpretation 2: a **has scope over** every

- Scope is not uniquely determined either by left-to-right order, or by position in the parse tree.

- We therefore need other mechanisms to ensure that the ambiguity is reflected in the LF assigned to S.

Scope ambiguity, continued

The number of interpretations grows exponentially with the number of scope operators:

Every student at **some** university has access to **a** laptop.

1. Not necessarily same laptop, not necessarily same university

$\forall x(stud(x) \wedge \exists y(univ(y) \wedge at(x,y)) \rightarrow \exists z(laptop(z) \wedge have\_access(x,z)))$

2. Same laptop, not necessarily same university

$\exists z(laptop(z) \wedge \forall x(stud(x) \wedge \exists y(univ(y) \wedge at(x,y)) \rightarrow have\_access(x,z)))$

3. Not necessarily same laptop, same university

$\exists y(univ(y) \wedge \forall x((stud(x) \wedge at(x,y)) \rightarrow \exists z(laptop(z) \wedge have\_access(x,z))))$

4. Same university, same laptop $\exists y(univ(y) \wedge \exists z(laptop(z) \wedge \forall x((stud(x) \wedge at(x,y)) \rightarrow have\_access(x,z))))$

5. Same laptop, same university $\exists z(laptop(z) \wedge \exists y(univ(y) \wedge \forall x((stud(x) \wedge at(x,y)) \rightarrow have\_access(x,z))))$

where 4 & 5 are equivalent

Every student at **some** university does **not** have access to **a** computer.
$\rightarrow$ 18 interpretations

Coping with Scope: options

Enumerate all interpretations: Computationally unattractive!

Semantic Underspecification: Build LFs via syntax that **underspecify** the relative semantic scopes of the quantifiers

- Partial description of a FoL formula
- So Syntax-Tree:LF is 1:1, but the LF describes several FoL formulae and hence several interpretations

Sometimes the surrounding context will help us choose between interpretations:

Every student has access to a computer. It can be borrowed from the ITO.
($\Rightarrow$ a outscopes every)

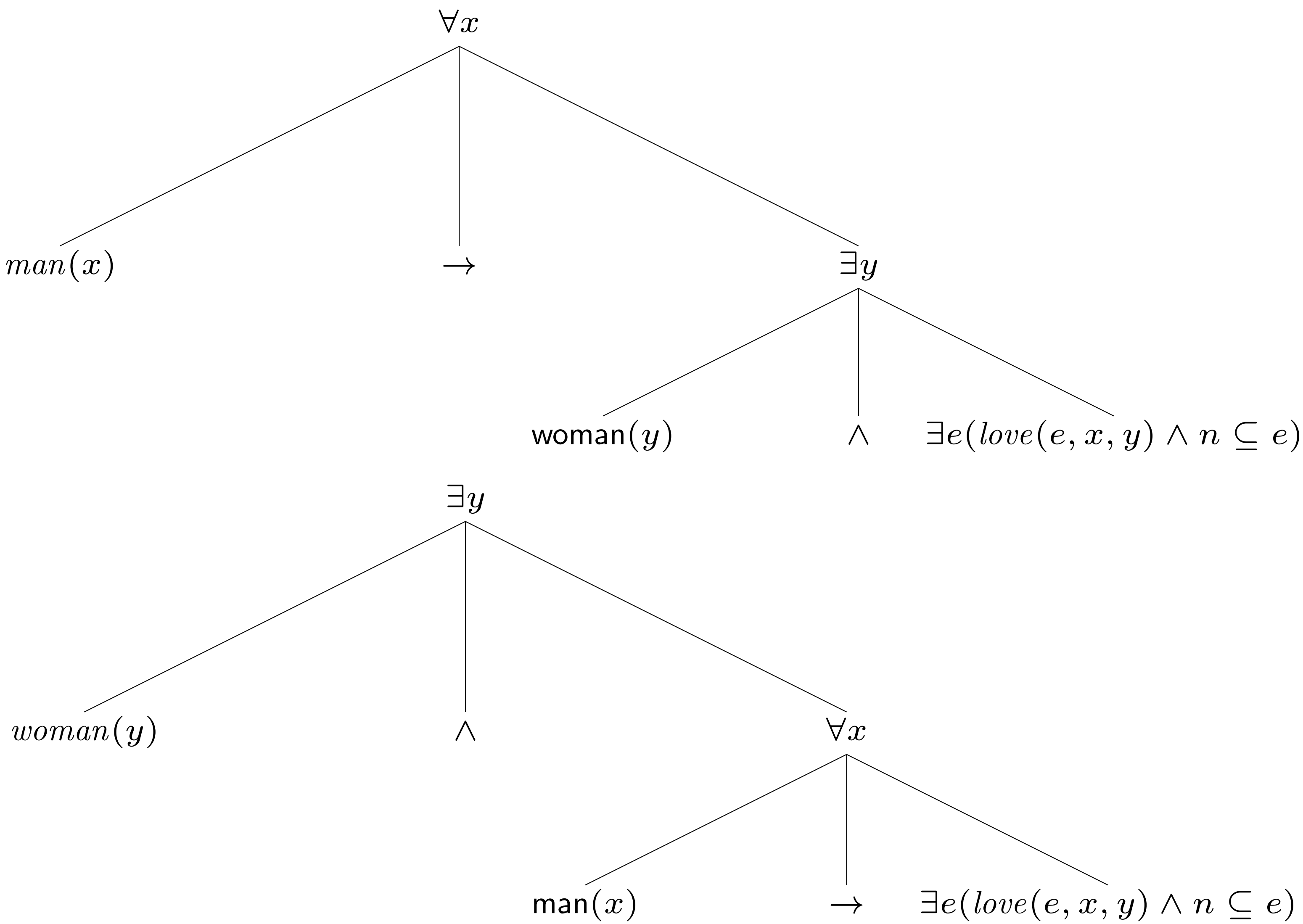
Semantic Underspecification

- The LF constructed in the grammar features:

  1. FoL bits
  2. constraints on how they can combine into an FoL formula

- The constraints are satisfied by more than one FoL formula.

A Picture showing common bits and different bits



Technique

- Label nodes of the tree: $l_1, l_2 \ldots$
- Supply constraints on what FoL expressions appear at those labels

Every man loves a woman.

Ignoring $\exists e$ and $n \subseteq e$...

$$\begin{array}{ll} l_1: & \forall x(h_2 \rightarrow h_3) \\ l_2: & man(x) \\ l_3: & love(e,x,y) \\ l_4: & \exists y(h_4 \wedge h_5) \\ l_5: & woman(y) \\ \multicolumn{2}{l}{h_2 =_q l_2, h_4 =_q l_5} \end{array}$$

Resolving Scope

$$
\begin{array}{ll}
l_1: & \forall x(h_2 \rightarrow h_3) \\
l_2: & man(x) \\
l_3: & love(e, x, y) \\
l_4: & \exists y(h_4 \wedge h_5) \\
l_5: & woman(y) \\
\multicolumn{2}{l}{h_2 =_q l_2, h_4 =_q l_5}
\end{array}
$$

- All hs must equal a (unique) l; no free variables

- So there are two solutions:

  $\exists$ **outscopes** $\forall$**:** $h_2 = l_2, h_4 = l_5, h_3 = l_3 h_5 = l_1$
  $\forall$ **outscopes** $\exists$**:** $h_2 = l_2, h_4 = l_5, h_3 = l_1, h_5 = l_3$

- LF construction via the grammar must now λ-abstract labels, as well as predicates, arguments to predicates etc.

Summary

- Syntax guides semantic composition in a systematic way.
- Lambda expressions facilitate the construction of compositional semantic interpretations off the syntax tree.
  - Associate each word with a λ-term
  - Within the grammar, say which daughter is the functor.
- However, semantic scope ambiguities suggest that not all semantic ambiguities should surface as syntactic ambiguities within the grammar.
- There are solutions to this that exploit semantic underspecification.

Next Lecture: Semantic Role Labelling