6.863J Natural Language Processing
Lecture 18: the meaning of it all, #4
(or #42)

Instructor: Robert C. Berwick
berwick@ai.mit.edu

The Menu Bar

- Administrivia:
  - Lab 4a out April 14 – last lab before final project

Agenda:

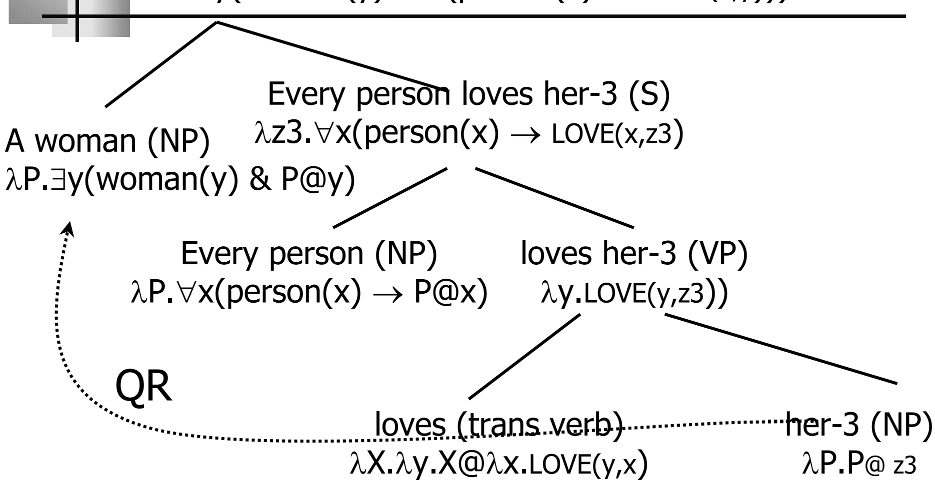
- Scoping ambiguities & computation - solutions
- Quantifier raising (QR)
- Cooper storage
- Keller storage
- "Hole" semantics
- Prelude to discourse representation theory

Montague's approach (& other current linguistic theory)

- Rule of Quantifier raising – like moving other phrases
- Landing site in position at head of Sbar (function or operator position)
- Combine with indexed pronoun (alternatively: empty element or trace) instead of quantifying NP
- When placeholder has moved high enough in tree to give the scope we need, replace by quantifying NP

Example: every person loves a woman (Sbar)

$\exists y(woman(y) \& \forall x(person(x) \rightarrow \text{LOVE}(x,y)))$



Why do we have to solve this?

- Readings aren't always logically independent
- Direct construction doesn't give us the right ambiguities
- Example (demo):

  every customer in a restaurant eats a big kahuna burger
- forall A ((exists B (restaurant(B) & in(A,B)) & customer(A)) -> exists C ((big(C) & (kahuna(C) & burger(C))) & eat(A,C)))

Here they all are…

1. forall A ((exists B (restaurant(B) & in(A,B)) & customer(A)) > exists C ((big(C) & (kahuna(C) & burger(C))) & eat(A,C)))
2. forall A ((in(A,B) & customer(A)) > exists C (restaurant(C) & exists D ((big(D) & (kahuna(D) & burger(D))) & eat(A,D))))
3. forall A ((in(A,B) & customer(A)) > exists C (restaurant(C) & exists D ((big(D) & (kahuna(D) & burger(D))) & eat(A,D))))
4. forall A ((in(A,B) & customer(A)) > exists C ((big(C) & (kahuna(C) & burger(C))) & exists D (restaurant(D) & eat(A,C))))
5. exists A ((big(A) & (kahuna(A) & burger(A))) & forall B ((exists C (restaurant(C) & in(B,C)) & customer(B)) > eat(B,A)))
6. exists A (restaurant(A) & forall B ((in(B,A) & customer(B)) > exists C ((big(C) & (kahuna(C) & burger(C))) & eat(B,C))))
7. exists A ((big(A) & (kahuna(A) & burger(A))) & forall B ((in(B,C) & customer(B)) > exists D (restaurant(D) & eat(B,A))))
8. exists A (restaurant(A) & exists B ((big(B) & (kahuna(B) & burger(B))) & forall C ((in(C,A) & customer(C)) > eat(C,B))))
9. exists A ((big(A) & (kahuna(A) & burger(A))) & exists B (restaurant(B) & forall C ((in(C,B) & customer(C)) > eat(C,A))))

Montague approach

- Idea of having a 'dummy' semantic rep that we use when needed is basically right…
- But… way it is used here is not smart from a modular engineering or computational design
- Don't want to futz w/ grammar – only want to add on this combinatory mechanism to existing grammars
- Storage methods – move the QR idea from syntax to semantics
- Cooper storage & Keller storage

Cooper storage

History: cf W. Woods and Lunar system

Key ideas:

- Associate each node of parse tree with a store
- Store contains core semantic rep together w/ quantifiers associated w/ nodes lower in the tree
- After sentence is parsed, store is used to generate scoped representations
- Order in which store is retrieved determines the different scopings (cf also for PP attachment…)

Formally stores

- A store is an *n*-place sequence
- Stores are represented by angle brackets < and >
- The first item of the sequence is the core semantic representation
- Subsequent elements are pairs (β, i) where β is the semantic representation of an NP (that is, another lambda expression) and i is an index
- An index is a label that picks out a free variable in the core semantic representation

Use of the store

- Quantified Noun phrases can repackage the information that the store contains

More precisely:

Storage (Cooper)

If the store $<\phi,(\beta, j),\ldots,(\beta', k)>$ is a semantic representation for a quantified NP, then the store $< \lambda P.P@z_i, \phi,(\beta, j),\ldots(\beta', k)>$ where i is some unique index, is also a rep for that NP

Let's try it

- Every person loves a woman

Tree for this showing indices

Every person loves a woman (S)

$<\text{LOVE}(z6,z7), (\lambda P.\forall x(\text{person}(x) \rightarrow P@x), 6), (\lambda P.\exists y(\text{woman}(y)\&P@y),7)>$

loves a woman (VP)

$<\lambda u.\text{LOVE}(u,z7),(\lambda P.\exists y(\text{woman}(y)\&P@y),7)>$

Every person (NP)

$< \lambda Q.Q@z6,(\lambda P.\forall x(\text{person}(x) \rightarrow P@x), 6)>$

loves (trans verb)

$\lambda X.\lambda u.X@\lambda v.\text{LOVE}(u,v)$

a woman (NP)

$< \lambda Q.Q@ z7,(\lambda P.\exists y(\text{woman}(y)\&P@y),7)>$

Retrieval 1

- Want the ordinary scoped representation
- How do we get this?
  - Remove one of the indexed binding operators from the store
  - Combine it with the core representation
  - Result is a new core representation
  - Continue until store has just one element

Or precisely

- Retrieval:
  - Let $\sigma 1$ and $\sigma 2$ be (possibly empty) sequences of binding operators
  - If the store $<\phi, \sigma 1, (\beta, \ i\ ), \sigma 2>$ is associated with an expression of category S, then the store

  $< \beta @ \lambda z_i . \phi, \sigma 1, \sigma 2>$ is also associated with this expression

  Informally: pull out the indexed QP and apply

Let's see how it works

<LOVE(z6,z7),(λP.∀x(person(x) → P@x), 6),(λP.∃y(woman(y)&P@y),7)>

- Retrieval rule to this store, pull 1st quantifier out

< λP.∀x(person(x) → P@x)@ λz6. LOVE(z6,z7), (λP.∃y(woman(y)&P@y),7)>

- Beta-convert (lambda apply) to simplify:

< ∀x(person(x) → love(x,z7)), (λP.∃y(woman(y)&P@y),7)>

- Pull 2nd quantifier (the last one remaining)

<λP.∃y(woman(y)&P@y)@λz7.∀x(person(x) → love(x,z7))>

- Result:

< ∃y(woman(y)&∀x(person(x) → love(x,y))>

How do we get the other reading?

Are we ok?

- Cooper storage gives a lot of freedom
- Quantifiers retrieved in any order
- The only constraint is the use of co-indexed variables
- Is this too much rope?

Mia knows every owner of a hash bar

Nested NPs cause a problem

- Store:

$\langle Know(Mia, z2), (\lambda P.\forall y(owner(y)\&Of(y,z1)\rightarrow P@y),2),$
$(\lambda Q.\exists x(hashbar(x)\&Q@a),1)\rangle$

- Pull 2:

$\langle \forall y(owner(y)\&Of(y,z1)\rightarrow Know(Mia,y)),$
$\lambda Q.\exists x(hashbar(x)\&Q@a),1)\rangle$

$\langle \exists x(hashbar(x)\& \forall y(owner(y)\&Of(y,x) \rightarrow Know(Mia,y)))\rangle$

- Pull 1:

$\langle \exists x(hashbar(x)\&Know(Mia,z2)),$
$(\lambda P.\forall y(owner(y)\&Of(y,z1)\rightarrow P@y),2)\rangle$

$\langle \forall y(owner(y)\&Of(y,\underline{z1})\rightarrow \exists x(hashbar(x)\&Know(Mia,y)))\rangle$

???????

What to do?

- Allow stores to contain other stores
- Nesting structure of stores automatically tracks nesting of NPs
- Easy to implement (akin to some linguistic solutions: can't move NP 'too far')
- Keller storage: If the (nested) store $\langle \phi, \sigma \rangle$ is an interpretation for an NP, then the (nested) store $\langle \lambda P.P@z_i, (\langle \phi, \sigma \rangle, i)$, for some unique index i, is also an interpretation for this NP

Every owner…

Every owner of a hashbar (NP)

$\lambda P.P@z2,(<(\lambda P.\forall y(owner(y)\&Of(y,z1)\rightarrow P@y),(< \lambda Q.\exists x(hashbar(x)\&Q@x)>,1)>,2).$

Every(Det)

$<\lambda Q.\ \lambda P.\ \forall y\ (Q@y \rightarrow P@y)>$

owner of a hashbar (Nbar)

$<\lambda u.owner(u)\ \&\ of(u,\ z1),(< \lambda Q.\exists x(hashbar(x)\&Q@x)>,1)>$

owner (Noun)

$<\lambda x.owner(x)>$

of a hashbar (PP)

$< \lambda P.\lambda u.P@u\ \&\ of(u,\ z1),(< \lambda Q.\exists x(hashbar(x)\&Q@x)>,1)>$

Retrieval with nested storage

- The new retrieval rule:

  Let σ, $\sigma 1$, $\sigma 2$, be (possibly empty) sequences of binding operators

  If the (nested) store $<\phi, \sigma 1, (<\beta,\sigma>, i), \sigma 2>$ is an interpretation for an expression of category S, then $<\beta@\lambda z_i.\phi, \sigma 1, \sigma, \sigma 2>$ is too
- Ensures that any operators stored while processing β become accessible for retrieval only after β itself has been retrieved
- Overcomes problem with generating free variable readings

Reading 1

- Nested store:

<

know(Mia, z2)

(

<

λP.∀y(owner(y)&Of(y,z1)→P@y),

(

< λQ.∃x(hashbar(x)&Q@x)>,1

)

>, 2)

> Only one way to do retrieval:

<∃x(hashbar(x)& ∀y(owner(y)&Of(y,x) →Know(Mia,y)))>

Reading 2 – avoid storing nested NP 'a hashbar'

Every owner of a hashbar

(NP)

λP.P@z2, (<(λP.∀y(owner(y)& ∃x(hashbar(x)& of(z,x))→P@y)>,2)>

Every(Det)

<λQ. λP. ∀y (Q@y → P@y)>

owner of a hashbar (Nbar)

<λz.owner(z) & ∃ x (hashbar(x)& of(z,x))>

owner (Noun)

<λx.owner(x)>

of a hashbar (PP)

< λP.λz.P@z & ∃ x (hashbar(x)& of(z,x))>

Basic message

- Pushing quantifier on store is nondeterministic choice
- Use nested stores to deal with complex NPs

Quantifier store conclusions

- Original version isn't sufficiently constrained
- Causes spurious readings (in fact, logical nonsense)
- Cure: nested stores – only trivial changes to Cooper store
- Is it enough? Consider:

  One criminal knows every owner of a hash bar

Problem

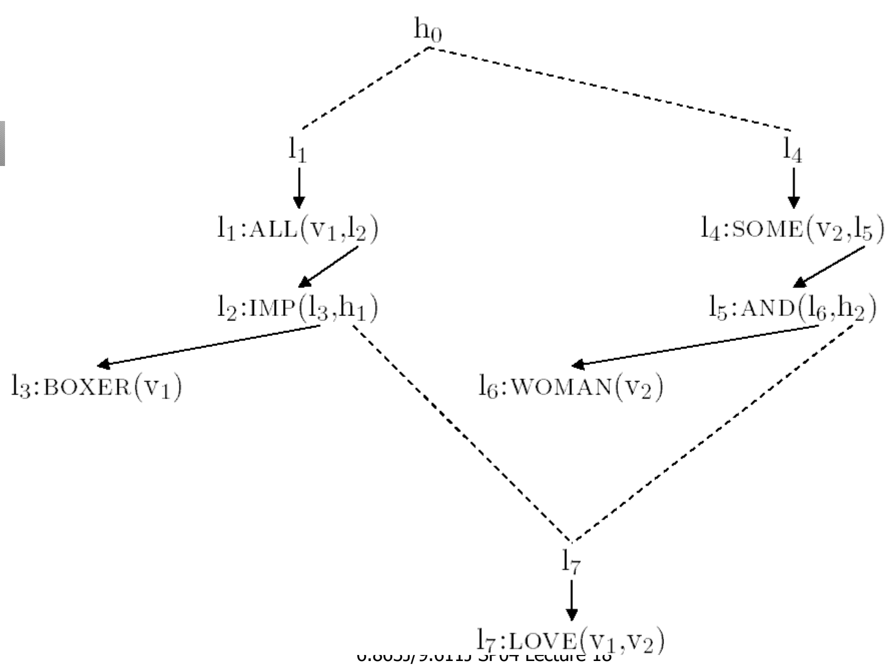
- Storage lets us represent possible combinations compactly, and gets 5 readings for this but…
- Doesn't let us <u>force</u> 'every owner' outscope 'a hash bar' while leaving subj-obj relation intact
- How do we <u>add</u> other constraints like this?
- Solution: <u>underspecified</u> constraint system – add constraints… how?
- What about negation? Storage doesn't handle this!

Hole semantics

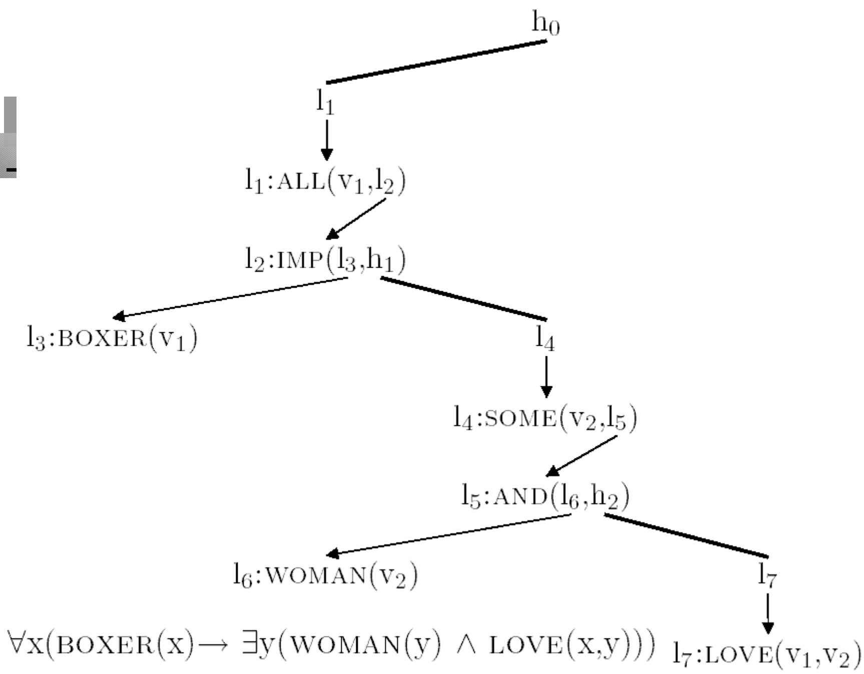
- Constraint satisfaction method

Every boxer loves a woman – UF:

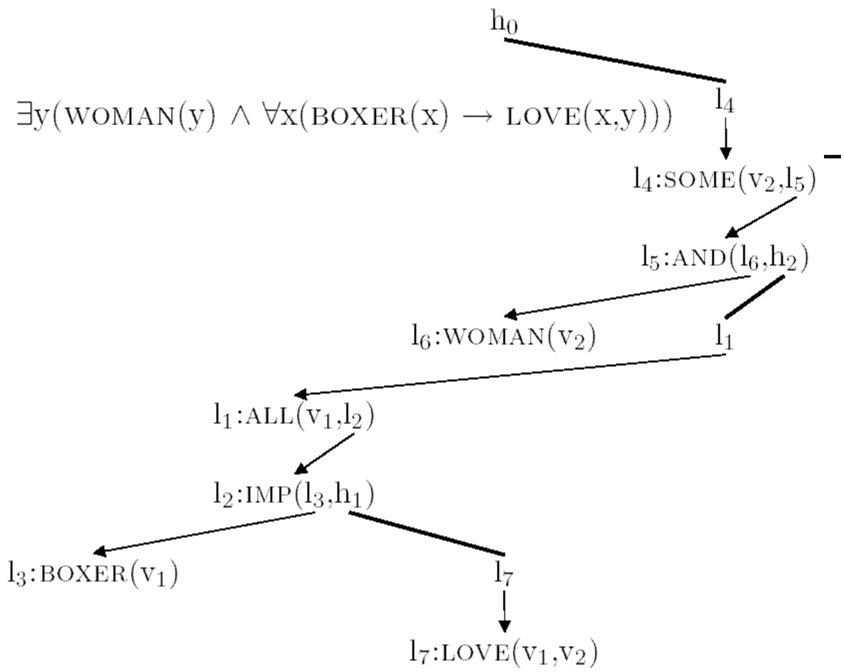
$$\exists l_1 \exists l_2 \exists v_1 (l_1{:}\textsc{all}(v_1,l_2) \wedge \exists l_3 \exists h_1 (l_2{:}\textsc{imp}(l_3,h_1) \wedge l_3{:}\textsc{boxer}(v_1) \wedge$$
$$\exists l_4 \exists l_5 \exists v_2 (l_4{:}\textsc{some}(v_2,l_5) \wedge \exists l_6 \exists h_2 (l_5{:}\textsc{and}(l_6,h_2) \wedge l_6{:}\textsc{woman}(v_2) \wedge$$
$$\exists l_7 (l_7{:}\textsc{love}(v_1,v_2) \wedge l_7 \leq h_1 \wedge l_7 \leq h_2 \wedge \exists h_0 (l_1 \leq h_0 \wedge l_4 \leq h_0))))))$$



6.863J/9.611J SP04 Lecture 18


h0
l1
l1:ALL(v1,l2)
l2:IMP(l3,h1)
l3:BOXER(v1)
l4
l4:SOME(v2,l5)
l5:AND(l6,h2)
l6:WOMAN(v2)
l7
∀x(BOXER(x)→ ∃y(WOMAN(y) ∧ LOVE(x,y)))
l7:LOVE(v1,v2)

$\exists$y(WOMAN(y) $\wedge$ $\forall$x(BOXER(x) $\rightarrow$ LOVE(x,y)))



Translation of a UF and a plugging to FOL:

$(\mathrm{h},P)^{\mathrm{uf2fol}} = (P(\mathrm{h}))^{\mathrm{uf2fol}}$ iff h is a hole
$(\mathrm{l},P)^{\mathrm{uf2fol}} = \exists \mathrm{v}(\mathrm{n},P)^{\mathrm{uf2fol}}$ iff l:SOME(v,n)
$(\mathrm{l},P)^{\mathrm{uf2fol}} = \forall \mathrm{v}(\mathrm{n},P)^{\mathrm{uf2fol}}$ iff l:ALL(v,n)
$(\mathrm{l},P)^{\mathrm{uf2fol}} = ((\mathrm{n},P)^{\mathrm{uf2fol}} \wedge (\mathrm{n}',P)^{\mathrm{uf2fol}})$ iff l:AND(n,n′)
$(\mathrm{l},P)^{\mathrm{uf2fol}} = $ C(v) iff l:C(v)

Basic UFs are defined as follows:

1. If l is a label, and h is a hole, then l≤h is a basic UF;
2. If l is a label, and n and n′ are nodes, then l:NOT(n), l:IMP(n,n′), l:AND(n,n′), l:OR(n,n′) are basic UFs;
3. If l is a label, t and t′ are terms, then l:EQ(t,t′) is a basic UF;
4. If l is a label, S is a symbol in the SRL language with arity n, and t_1 $\ldots t_n$ are terms, then l:S,t_1,…,t_n) is a basic UF.
5. If l is a label, v a metavariable, and n a hole or label, then l:SOME(v,n) and l:ALL(v,n) are basic UFs.
6. Nothing else is a basic UF.

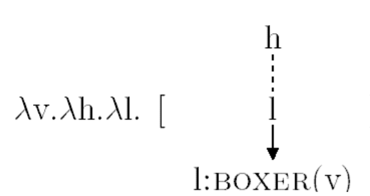
Basic UFs are defined as follows:

1. If l is a label, and h is a hole, then l≤h is a basic UF;
2. If l is a label, and n and n′ are nodes, then l:NOT(n), l:IMP(n,n′), l:AND(n,n′), l:OR(n,n′) are basic UFs;
3. If l is a label, t and t′ are terms, then l:EQ(t,t′) is a basic UF;
4. If l is a label, S is a symbol in the SRL language with arity n, and t_1 $\ldots t_n$ are terms, then l:S,t_1,…,t_n) is a basic UF.
5. If l is a label, v a metavariable, and n a hole or label, then l:SOME(v,n) and l:ALL(v,n) are basic UFs.
6. Nothing else is a basic UF.

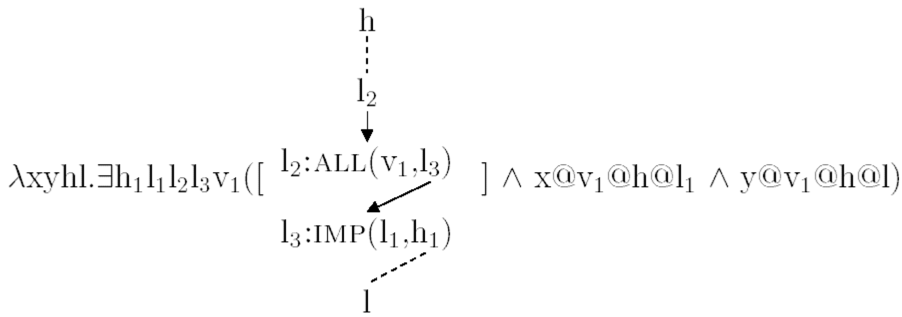
2. If ϕ is a UF, and n is a node then $\exists n \phi$ is a UF;
3. If ϕ is a UF, and v is a meta-variable then $\exists v \phi$ is a UF;
4. If ϕ and ψ are UFs,then $(\phi \wedge \psi)$ is a UF;
5. Nothing else is a UF.

λv.λh.λl.(BOXER(l,v) $\wedge$ l$\leq$h).

Drawn as a tree:

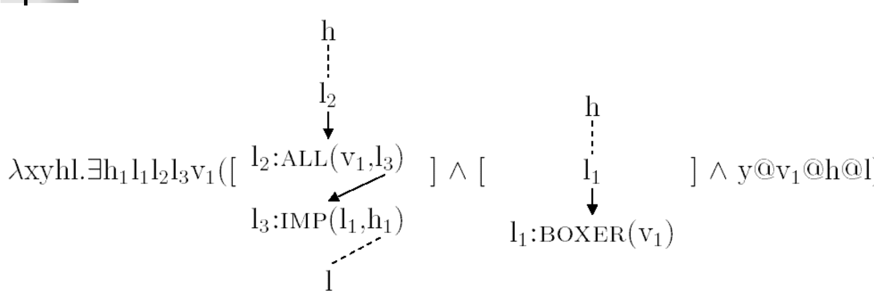


Every

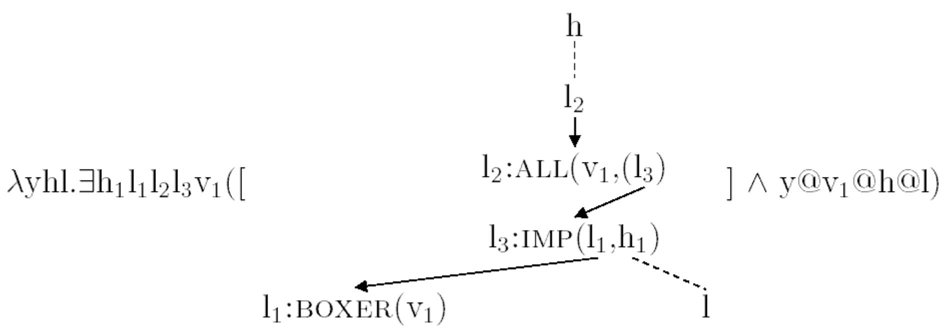


Every boxer

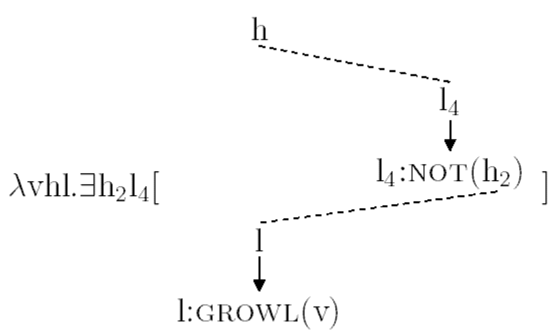
Two separate trees…



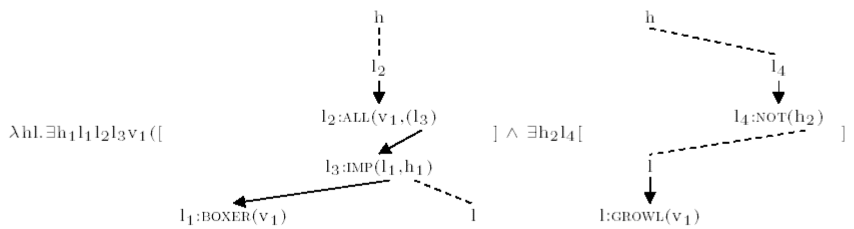
Now combine them…



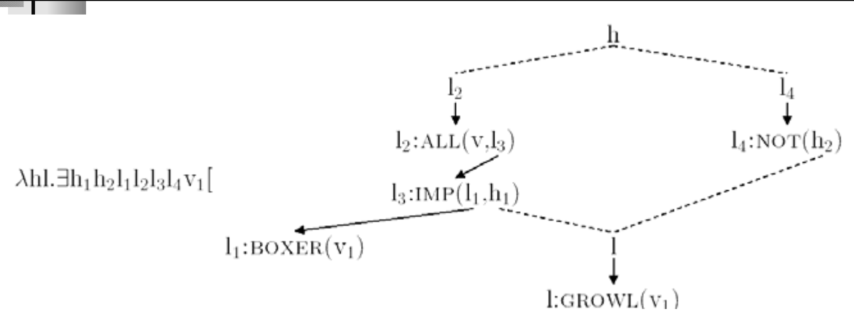
Does not growl



Every boxer does not growl



'every boxer does not growl'



Now we need a 'plugging' algorithm…

Why underspecify?

- One criminal knows the owner of every hash bar
- What do storage methods do?
- What does UF do? (or rather, can do)?

Meaning of meaning, redux

- How can we automate process of associating semantic representations w/ expressions of natural language?
- How can we use semantic representations to automate drawing inferences?

Why compositionality?

- a human being can understand a possibly *infinite* number of sentences never heard before (namely by constructing their meaning from a *finite* set of rules and a *finite* set of known lexical meanings).
- Also, a compositional account of meaning suggests a plausible explanation of why we perceive a connection *in meaning* between sentences that share syntactic parts