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)

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

A woman (NP) \(\lambda P.\exists y(\text{woman}(y) \& P@y)\)

Every person (NP) \(\lambda P.\forall x(\text{person}(x) \rightarrow P@x)\)

loves her-3 (VP) \(\lambda y.\text{LOVE}(y,z3)\)

QR

loves (trans. verb) \(\lambda X.\lambda y.X@\lambda X.\text{LOVE}(y,x)\)

her-3 (NP) \(\lambda P.P@ z3\)
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
  - \( \forall A \left( \exists B \left( \text{restaurant}(B) \land \text{in}(A,B) \right) \land \text{customer}(A) \right) \rightarrow \exists C \left( \left( \text{big}(C) \land \left( \text{kahuna}(C) \land \text{burger}(C) \right) \right) \land \text{eat}(A,C) \right) \)

Here they all are...

1. \( \forall A \left( \exists B \left( \text{restaurant}(B) \land \text{in}(A,B) \right) \land \text{customer}(A) \right) \rightarrow \exists C \left( \left( \text{big}(C) \land \left( \text{kahuna}(C) \land \text{burger}(C) \right) \right) \land \text{eat}(A,C) \right) \)
2. \( \forall A \left( \left( \text{in}(A,B) \land \text{customer}(A) \right) \rightarrow \exists C \left( \text{restaurant}(C) \land \exists D \left( \left( \text{big}(D) \land \left( \text{kahuna}(D) \land \text{burger}(D) \right) \right) \land \text{eat}(A,D) \right) \right) \right) \)
3. \( \forall A \left( \left( \text{in}(A,B) \land \text{customer}(A) \right) \rightarrow \exists C \left( \text{restaurant}(C) \land \exists D \left( \left( \text{big}(D) \land \left( \text{kahuna}(D) \land \text{burger}(D) \right) \right) \land \text{eat}(A,D) \right) \right) \right) \)
4. \( \forall A \left( \left( \text{in}(A,B) \land \text{customer}(A) \right) \rightarrow \exists C \left( \left( \text{big}(C) \land \left( \text{kahuna}(C) \land \text{burger}(C) \right) \right) \land \text{exists D} \left( \text{restaurant}(D) \land \text{eat}(A,C) \right) \right) \right) \)
5. \( \exists A \left( \left( \text{big}(A) \land \left( \text{kahuna}(A) \land \text{burger}(A) \right) \right) \land \forall B \left( \left( \exists C \left( \text{restaurant}(C) \land \text{in}(B,C) \land \text{customer}(B) \right) \right) \rightarrow \text{eat}(B,A) \right) \right) \)
6. \( \exists A \left( \text{restaurant}(A) \land \forall B \left( \left( \text{in}(B,A) \land \text{customer}(B) \right) \rightarrow \exists C \left( \left( \text{big}(C) \land \left( \text{kahuna}(C) \land \text{burger}(C) \right) \right) \land \text{eat}(B,C) \right) \right) \right) \)
7. \( \exists A \left( \left( \text{big}(A) \land \left( \text{kahuna}(A) \land \text{burger}(A) \right) \right) \land \forall B \left( \left( \text{in}(B,C) \land \text{customer}(B) \right) \rightarrow \exists D \left( \text{restaurant}(D) \land \text{eat}(B,A) \right) \right) \right) \)
8. \( \exists A \left( \text{restaurant}(A) \land \exists B \left( \left( \text{big}(B) \land \left( \text{kahuna}(B) \land \text{burger}(B) \right) \right) \land \forall C \left( \left( \text{in}(C,A) \land \text{customer}(C) \right) \rightarrow \text{eat}(C,B) \right) \right) \right) \)
9. \( \exists A \left( \left( \text{big}(A) \land \left( \text{kahuna}(A) \land \text{burger}(A) \right) \right) \land \exists B \left( \text{restaurant}(B) \land \forall C \left( \left( \text{in}(C,B) \land \text{customer}(C) \right) \rightarrow \text{eat}(C,A) \right) \right) \right) \)
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 $(\beta, i)$ where $\beta$ 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 $<\phi, (\beta, j), ..., (\beta', k)>$ is a semantic representation for a quantified NP, then the store $<\lambda P. P@z_i, \phi, (\beta, j), ..., (\beta', k)>$ 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)
<LOVE(z6,z7), (λ.P.∀x(person(x) → P@x), 6), (λ.P.∃y(woman(y)&P@y),7)>

loves a woman (VP)

Every person (NP)
<λ.u.LOVE(u,z7),(λ.P.∃y(woman(y)&P@y),7)>
<λ.Q@z6,(λ.P.∀x(person(x) → P@x), 6)>

loves (trans verb)
λ.X.λ.u.X@λ.v.LOVE(u,v)

a woman (NP)
<λ.Q@ z7,(λ.P.∃y(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

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

- Retrieval rule to this store, pull 1st quantifier out
  \(<\lambda P. \forall x \text{person}(x) \rightarrow P@x)@\lambda z6. \text{LOVE}(z6,z7), (\lambda P. \exists y \text{woman}(y) \& P@y), 7)>\)
- Beta-convert (lambda apply) to simplify:
  \(<\forall x (\text{person}(x) \rightarrow \text{love}(x,z7)), (\lambda P. \exists y \text{woman}(y) \& P@y), 7)>\)
- Pull 2nd quantifier (the last one remaining)
  \(<\lambda P. \exists y \text{woman}(y) \& P@y)@\lambda z7. \forall x (\text{person}(x) \rightarrow \text{love}(x,z7))>\)
- Result:
  \(<\exists y (\text{woman}(y) \& \forall x (\text{person}(x) \rightarrow \text{love}(x,y))>\)

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:**
  \[
  \langle \text{Know(Mia, z2), } \langle \lambda P. \forall y(\text{owner}(y) \& \text{Of(y, z1) } \rightarrow P@y), 2), \rangle, \langle \lambda Q. \exists x(\text{hashbar}(x) \& Q@a), 1) \rangle \rangle
  \]

- **Pull 2:**
  \[
  \langle \forall y(\text{owner}(y) \& \text{Of(y, z1) } \rightarrow \text{Know(Mia, y)}), \lambda Q. \exists x(\text{hashbar}(x) \& Q@a), 1) \rangle, \langle \exists x(\text{hashbar}(x) \& \forall y(\text{owner}(y) \& \text{Of(y, x) } \rightarrow \text{Know(Mia, y)})) \rangle
  \]

- **Pull 1:**
  \[
  \langle \exists x(\text{hashbar}(x) \& \text{Know(Mia, z2))}, \lambda P. \forall y(\text{owner}(y) \& \text{Of(y, z1) } \rightarrow P@y), 2rangle \rangle, \langle \forall y(\text{owner}(y) \& \text{Of(y, z1) } \rightarrow \exists x(\text{hashbar}(x) \& \text{Know(Mia, y)))) \rangle
  \]

---

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 \( \langle \lambda P.P@z_i(\langle \phi, \sigma \rangle, i) \rangle \), for some unique index \( i \), is also an interpretation for this NP
Every owner...

Every owner of a hashbar

\[ \lambda P. \forall y (\text{owner}(y) \& \text{of}(y, z1) \rightarrow P@y), (\langle \lambda Q. \exists x (\text{hashbar}(x) \& Q@x) >, 1 >, 2). \]

Every(Det)
\[ \langle \lambda Q. \lambda P. \forall y (Q@y \rightarrow P@y) > \]

owner of a hashbar (Nbar)
\[ \langle \lambda u. \text{owner}(u) \& \text{of}(u, z1), (\langle \lambda Q. \exists x (\text{hashbar}(x) \& Q@x) >, 1 >) \]

owner (Noun)
\[ \langle \lambda x. \text{owner}(x) > \]

of a hashbar (PP)
\[ \langle \lambda P. \lambda u. P@u \& \text{of}(u, z1), (\langle \lambda Q. \exists x (\text{hashbar}(x) \& Q@x) >, 1 >) \]

Retrieval with nested storage

- The new retrieval rule:
  Let \( \sigma, \sigma1, \sigma2 \), be (possibly empty) sequences of binding operators
  If the (nested) store \( \langle \phi, \sigma1, (\langle \beta, \sigma >, i), \sigma2 > \) is an interpretation for an expression of category S, then \( \langle \beta@\lambda z, \phi, \sigma1, \sigma, \sigma2 > \) is too
- Ensures that any operators stored while processing \( \beta \) become accessible for retrieval only after \( \beta \) itself has been retrieved
- Overcomes problem with generating free variable readings
Reading 1

- Nested store:

\[
\text{know}(\text{Mia}, z_2) \\
( \\
\quad < \quad \lambda P. \forall y (\text{owner}(y) \& \text{Of}(y, z_1) \rightarrow P@y), \\
( \quad < \quad \lambda Q. \exists x (\text{hashbar}(x) \& Q@x), 1 \\
\quad >, 2) \\
> \text{Only one way to do retrieval:} \\
< \exists x (\text{hashbar}(x) \& \forall y (\text{owner}(y) \& \text{Of}(y, x) \rightarrow \text{Know}(\text{Mia}, y))) >
\]

Reading 2 – avoid storing nested NP ‘a hashbar’

- Every owner of a hashbar

\[
\lambda P. P@z_2, (<(\lambda P. \forall y (\text{owner}(y) \& \exists x (\text{hashbar}(x) \& \text{of}(z, x)) \rightarrow P@y)>, 2)>
\]

- Every (Det)

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

- Owner of a hashbar (Nbar)

\[
<\lambda z. \text{owner}(z) \& \exists x (\text{hashbar}(x) \& \text{of}(z, x))>
\]

- Owner (Noun)

\[
<\lambda x. \text{owner}(x)>
\]

- Of a hashbar (PP)

\[
<\lambda z. \lambda P. P@z \& \exists x (\text{hashbar}(x) \& \text{of}(z, x))>
\]
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 force ‘every owner’ outscope ‘a hash bar’ while leaving subj-obj relation intact
- How do we add other constraints like this?
- Solution: underspecified constraint system – add constraints... how?
- What about negation? Storage doesn’t handle this!

Hole semantics

- Constraint satisfaction method
Every boxer loves a woman – UF:

\[\exists v_1 \exists v_2 (l_1: \text{ALL}(v_1, l_2) \land \exists h_1(l_2: \text{IMP}(l_3, h_1) \land l_3: \text{BOXER}(v_1)) \land \exists v_3 v_4 (l_4: \text{SOME}(v_2, l_5) \land \exists h_2(l_5: \text{AND}(l_6, h_2) \land l_6: \text{WOMAN}(v_2)) \land \exists l_7(l_7: \text{LOVE}(v_1, v_2) \land l_7 \leq h_1 \land l_7 \leq h_2 \land \exists l_0(l_1 \leq l_0 \land l_4 \leq l_0))))\]
∀x(BOXER(x) → ∃y(WOMAN(y) ∧ LOVE(x,y)))  \text{ l₇:LOVE(v₁,v₂)}
Translation of a UF and a plugging to FOL:

\[(h, P)^{uf2fo}_{\text{uf2fol}} = (P(h))^{uf2fo}_{\text{uf2fol}} \text{ iff } h \text{ is a hole}\]
\[(l, P)^{uf2fo}_{\text{uf2fol}} = \exists v(n, P)^{uf2fo}_{\text{uf2fol}} \text{ iff } l:\text{SOME}(v, n)\]
\[(l, P)^{uf2fo}_{\text{uf2fol}} = \forall v(n, P)^{uf2fo}_{\text{uf2fol}} \text{ iff } l:\text{ALL}(v, n)\]
\[(l, P)^{uf2fo}_{\text{uf2fol}} = (\langle n, P \rangle^{uf2fo}_{\text{uf2fol}} \land \langle n', P \rangle^{uf2fo}_{\text{uf2fol}}) \text{ iff } l:\text{AND}(n, n')\]
\[(l, P)^{uf2fo}_{\text{uf2fol}} = c(v) \text{ iff } l:c(v)\]
Basic UF's are defined as follows:

1. If $l$ is a label, and $h$ is a hole, then $l \leq h$ is a basic UF;

2. If $l$ is a label, and $n$ and $n'$ are nodes, then $\neg n$, $\land(n,n')$, $\lor(n,n')$ are basic UF's;

3. If $l$ is a label, $t$ and $t'$ are terms, then $\equiv(t,t')$ 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 $l:S,t_1,\ldots,t_n$ is a basic UF.

5. If $l$ is a label, $v$ a metavariable, and $n$ a hole or label, then $\exists v,(v,n)$ and $\forall v,(v,n)$ are basic UF's.

6. Nothing else is a basic UF.
2. If $\phi$ is a UF, and $n$ is a node then $\exists n \phi$ is a UF;

3. If $\phi$ is a UF, and $v$ is a meta-variable then $\exists v \phi$ is a UF;

4. If $\phi$ and $\psi$ are UF's, then $(\phi \land \psi)$ is a UF;

5. Nothing else is a UF.

\[ \lambda v.\lambda h.\lambda l. (\text{boxer}(l,v) \land l \leq h). \]

Drawn as a tree:

\[ \lambda v.\lambda h.\lambda l. \]

\[ \text{l:boxer(v)} \]
Every boxer

Two separate trees...

Now combine them...
\( \lambda y h. \exists l_1 l_2 l_3 v_1 ( [ l_2 : \text{ALL}(v_1, (l_3) ] \land y @ v_1 \odot h \odot l ) \)

\( l_3 : \text{IMP}(l_1, h_1 ) \)

\( l_1 : \text{BOXER}(v_1) \)

Does not growl

\( \lambda y h. \exists l_4 ( [ l_4 : \text{NOT}(h_2) ] \)

\( l_4 : \text{GROWL}(v) \)
Every boxer does not growl

\[ \lambda h. \exists h_1 h_2 h_3 v_1 ( h \land h_2 \land \lambda v_1 ( h_2 \land h_3 \land \lambda \{ \land \exists h_2 \} \land \lambda h_4 ( h_4 \land \text{BOXER}(v_1) \iff \text{GROWL}(v_1) ) ) ) \]

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.