1 Syntax and semantics with variables

- In the “standard semantics” exemplified by Heim & Kratzer, bound pronouns and traces are treated as variables. Specifically, they are expressions in $\langle e \rangle$ that are dependent on an assignment to variables. Or, putting it differently, they denote functions from an assignment function to an individual.

\[ [\text{he}_n]^g = [t_n]^g = g(n) \]

Notice that, under this view, there are infinitely many expressions pronounced “he,” infinitely many traces, and so on.

- Syntactically, we observe certain constraints on ‘binding.’

\[ (2) \text{ Weak Crossover} \]
\[ \text{a. } [\text{Every man}]_k \text{ loves his}_k \text{ dog.} \]
\[ \text{b. } * \text{ His}_k \text{ dog loves } [\text{every man}]_k. \]

\[ (3) \text{ Principle B} \]
\[ \text{a. } \text{John loves himself.} \]
\[ \text{b. } * \text{ John}_k \text{ loves him}_k. \]

As usually described, these are constraints on the relative positions of the variable and its noun phrase “binder” at a certain level of syntactic representation.

- But under the standard treatment of traces and pronouns—plus the now standard treatment of noun phrases as GQs in $\langle \langle e, t \rangle, t \rangle$—there really is no sense in which the quantificational noun phrase binds the pronoun.

Rather, binding is accomplished by covertly ‘abstracting’ over the index associated with the pronoun/trace.
(4) \[ t_3 \text{ loves his}_3 \text{ dog } \]

a. Prior to covert abstraction:
   \[ = \text{love}'(g(3), \nu y[\text{dog}(y) \land \text{poss}(g(3), y)]) \in (t) \]

b. After covert abstraction:
   \[ = \lambda x.\text{love}'(x, \nu y[\text{dog}(y) \land \text{poss}(x, y)]) \in (e, t) \]

• Correspondingly, the pretheoretical notion of ‘binding’ used in our descriptions finds only very indirect expression in the standard semantic, as articulated by a certain German duo, in what I’ll dub the relation of ‘binding’:

– Noun phrase B binds pronoun or trace V iff:
  1. B c-commands and is co-indexed with V at a certain level of syntax.
  2. V and the trace of B are sem-bound by the same thing.
  3. L sem-binds pronoun or trace X with index \( n \) iff L’s sister at LF is the smallest subtree whose semantic interpretation varies with a choice between variable assignments differing only in their assignment to \( n \).

Jacobson objects to the representational abandon of this reconstruction. It refers to (a) surface syntax, (b) LF syntax, and (c) interpretation.

2 Variable free semantics

• The idea of variable-free semantics is to do away with variable assignments, and do everything directly with functions—or equivalently, with \( \lambda \)-bound variables. Any variable bound by a \( \lambda \) is dispensible.

• Jacobson’s claim is that the result is not only just as adequate empirically, but metatheoretically more desirable, because:
  1. A single pronoun, rather than infinitely many
  2. All the action is local
  3. Doesn’t require “reconstruction” to get local binding of variables
  4. . . .
3 The category of pronouns

• Ordinary pronoun:
  1. Syntactic Category: NP
     Gloss: A category $A^B$ has the distribution of an $A$, but the semantic type of an $A/B$.
  2. Semantic Value: $\lambda x[x]
     Gloss: Identity function over individuals.

4 Combining verbs with pronouns

• How do verbs combine with pronouns?

Geach the verb—just like Jacobson (1990) did Raising verbs with bare AP complements—but this time, relative to our new syntactic category:

\[
G[B/A : f] \equiv B^C/A^C : \lambda v\lambda c. f(v(c))
\]

Example, with intransitive verb

a. dance = $S\backslash NP : dance'$

b. $G(\text{dance}) = S^{NP}\backslash NP^{NP} : \lambda V(\langle e,e \rangle)\lambda x(\langle e \rangle). dance'(V x)$

c. He $G(\text{dance}) = [NP^{NP} : \lambda x.x] [S^{NP}\backslash NP^{NP} : \lambda V(\langle e,e \rangle)\lambda x(\langle e \rangle). dance'(V x)]$
   $= S^{NP} : \lambda x. dance'(\lambda x.x(x)) \equiv \lambda x. dance'(x) \equiv dance'$

What does this rule do?
In the basic case, it just turns $f$ into a function that swallows pronouns and returns $f$. We'll see some nonbasic cases below.

(7) Example, with transitive verb

a. smack = $(S\backslash NP) / NP : smack'$

b. $G(\text{smack}) = (S\backslash NP)^{NP} / NP^{NP} : \lambda V \lambda x.smack'(V x) \equiv \lambda V \lambda x\lambda y.smack'(V x)(y)$

c. $G(\text{smack}) \ \text{him} = (S\backslash NP)^{NP} : \lambda x\lambda y.smack'(x)(y) \equiv smack'$
• The basic idea is, G accomplishes what the standard theory does by relativizing everything to variable assignments—thereby ‘passing indices upwards,’

• Notice, this means that, while sentences without free pronouns denote in \langle t \rangle, sentences with free pronouns do not. Instead, they denote predicates. We then derive a truth-evaluable statement ‘in our heads’ by supplying an individual to apply the predicate to.

Question: Is this any stranger than assuming that, ‘in our heads,’ we supply an assignment function that makes a sentence in \langle t \rangle informative?

• Clarificatory Questions

Why not just let pronouns be NP/NP and have the verb compose over the pronoun?

- NP/NP  S\NP  \Rightarrow_B  S/NP

Answer:

1. Need a type that distinguishes pronouns (NP\NP) from a noun phrase modifier (NP/NP)
2. We don’t want “he danced John” to mean ‘John danced,’ or “Al smacked it the table” to mean ‘Al smacked the table.’

The new category and its accoutrements are designed to exempt pronouns from what would otherwise be the distributional consequences of their semantic type. By giving them a special category, we can rig the grammar so that, despite their type, they have the distribution of NPs.

• Pronouns in adjuncts are handled by type-lifting the verb over any modifier.

5 Binding

• Binding is achieved by the Z combinator

\[
(8) \quad Z[ (B/NP)/A : f ] = (B/NP)/A^{NP} : \lambda g \lambda x. f(gx)(x)
\]

\[
(9) \quad \text{For example:}
\]

a. criticizes: (S\NP)/NP : criticize'

b. z(criticizes): (S\NP)/NP^{NP} : \lambda g \lambda x. criticize'(gx)(x)

This has two effects:
1. It makes a function over domain \( \langle e \rangle \), the type of plain NPs, into one over domain \( \langle e, e \rangle \), the type of pronouns. That’s the “\( \lambda g \)“ in (8) and (9b).

2. It covalues two arguments, the \( x \) variables in (8) and (9b).

These variables correspond to the ‘logical subject’ and the argument of the ‘logical object’ pronoun.

Notice, whenever \( g \) is just the identity function, the effect of \( Z \) is just the reflexivization of its operand: \( Z(love) \) is the set of self-lovers.

- Thus \( Z \) is accomplishing at the verb level what the standard semantics does at the sentence level by abstracting over a certain index, when the sentence contains two instances of that index. That’s the idea.

- Of course we have to generalize \( Z \) to functions with more than two arguments:

\[
\begin{align*}
\text{(10) a. } & Z[ ((B/NP)/ \ldots )A ] = ((B/NP)/ \ldots )A^\text{NP} \\
\text{b. } & Zf = \lambda g[ \lambda d_1 \ldots \lambda d_n[ \lambda x \lambda f . f(gx)(d_1) \ldots (d_n)(x) ] ]
\end{align*}
\]

- A case like (11) is handled by applying \( Z \) twice to the embedding verb, making it into a function over something with the semantic type of a transitive verb.

\[
\begin{align*}
\text{(11) } & \text{Every man}_{k} \text{ thinks that the woman who loves him}_{k} \text{ hates his}_{k} \text{ mother.}
\end{align*}
\]

a. \text{the woman who loves him hates his mother} \sim \lambda x \lambda y . \text{hate}^\prime(\text{the woman who loves } y, \text{the mother of } x) \\

b. \text{thinks} \sim \text{think}^\prime \equiv \lambda \Phi \lambda s . \text{think}^\prime(\Phi)(s) \\

c. \text{Z(think}^\prime) \equiv \lambda g \lambda x . \text{think}^\prime(gx)(x) \\

d. \text{Z(Z(think}^\prime)) \equiv \lambda f \lambda z . [ \lambda g \lambda x . \text{think}^\prime(gx)(x)](fz)(z) = \lambda f \lambda z . [\text{think}^\prime(fzz)(z)] \\

e. \text{Z(Z(think}^\prime)) [\lambda x \lambda y . \text{hate}^\prime(\text{the woman who loves } y, \text{the mother of } x)] \\
\text{ i. } = \lambda f \lambda z . [\text{think}^\prime(fzz)(z)][\lambda x \lambda y . \text{hate}^\prime(\text{the woman who loves } y, \text{the mother of } x)] \\
\text{ ii. } = \lambda z . [\text{think}^\prime(\lambda x \lambda y . \text{hate}^\prime(\text{the woman who loves } y, \text{the mother of } x))zz)(z)] \\
\text{ iii. } = \lambda z . [\text{think}^\prime(\text{hate}^\prime(\text{the woman who loves } z, \text{the mother of } z)))(z)] \\

f. \text{every man} \sim \lambda P . \forall v[\text{man}^\prime(v) \rightarrow Pv]
g. Every man \(Z(Z(\text{thinks}))\) that the woman who loves him hates his mother 
\[\lambda P.\forall v[\text{man}'(v) \rightarrow P v][\lambda z.[\text{think}'(\text{the\.woman\.who\.loves}(z), \text{the\.mother\.of}(z))(z)](v)]\]

i. \[= \forall v[\text{man}'(v) \rightarrow \lambda z.[\text{think}'(\text{the\.woman\.who\.loves}(z), \text{the\.mother\.of}(z))(z)](v)](v)]\]

ii. \[= \forall v[\text{man}'(v) \rightarrow [\text{think}'(\text{the\.woman\.who\.loves}(v), \text{the\.mother\.of}(v))(v)](v)]\]

6 Weak Crossover

- Recall the definition of Curry’s Substitution combinator, which Steedman (following Szabolcsi) uses for parasitic gaps:

\[(12) \quad ((S f) g)x \equiv f(x(gx))\]

Thus: \(S f \equiv \lambda g \lambda x.f(x(gx))\)

Now juxtaposing \(S\) and \(Z\):

\[(13) \quad \begin{align*}
  a. & \quad Z f \equiv \lambda g \lambda x.f(gx)(x) \\
  b. & \quad S f \equiv \lambda g \lambda x.f(x(gx))
\end{align*}\]

Notice anything? This is why Jacobson calls her combinator “Z.”

Of course \(Z\) has only a very narrow range of possible effects, because of the restriction on what syntactic categories it can apply to, and what sorts of syntactic categories it outputs.

- Jacobson models Weak Crossover by excluding the combinator \(S\)—or rather, an order-variant \(S\) such that \(\overline{S} f \equiv \lambda x \lambda g.f(x(gx))\)—from the toolbox of lexical ‘type-shifts.’

\[(14) \quad \text{Every man loves his dog, derived using } Z\]

a. \(\text{his dog} \sim \lambda z iy[\text{dog}'(y) \land \text{poss}(z, y)]\)

b. \(\text{loves} \sim \text{love}'\)

c. \(Z(\text{love}') \equiv \lambda g \lambda x.\text{love}'(gx)(x)\)

d. \(Z(\text{love}') (\lambda x iy[\text{dog}'(y) \land \text{poss}(x, y)])\)

i. \[= \lambda g \lambda x.\text{love}'(gx)(x)(\lambda z iy[\text{dog}'(y) \land \text{poss}(z, y)])(x)](x)\]

ii. \[= \lambda x.\text{love}'(\lambda z iy[\text{dog}'(y) \land \text{poss}(z, y)](x))(x)\]
iii. \(= \lambda x.love'(\iota y[dog'(y) \land poss(x, y)])(x)\)

iv. \(= 'the set of things which love their dog'\)

e. \textit{every man} \(\leadsto \lambda P.\forall z[man'(z) \rightarrow Pz]\)

f. \textit{every man} \(Z(\text{loves}) \text{ his dog} \leadsto \forall z[man'(z) \rightarrow [\lambda x.love'(\iota y[dog'(y) \land poss(x, y)])(x)](z)] \)
\(= \forall z[man'(z) \rightarrow love'(\iota y[dog'(y) \land poss(z, y)])(z)]\)

\((15)\) \textbf{His dog loves every man, illegally derived with } \mathcal{S} \textbf{ }

a. \textit{loves} \(\leadsto love'\)

b. \(\mathcal{S}(love') \equiv \lambda x\lambda g.love'(x)(gx)\)

c. \textbf{Lift}(\mathcal{S}(love'))
   \begin{enumerate}
   \item \(\equiv \lambda Q\lambda g.\left[Q(\lambda y.[\mathcal{S}(love')](y))(g)\right]\)
   \item \(= \lambda Q\lambda g.\left[Q(\lambda y.[\lambda x\lambda f.love'(x)(fx)](y))(gy)\right]\)
   \item \(= \lambda Q\lambda g.\left[Q\lambda y.love'(y)(gy)\right]\)
   \end{enumerate}

d. \textit{every man} \(\leadsto \lambda P.\forall z[man'(z) \rightarrow Pz]\)

e. \textit{love every man} \(\leadsto \textbf{Lift}(\mathcal{S}(love'))[\lambda P.\forall z[man'(z) \rightarrow Pz]]\)
   \begin{enumerate}
   \item \(= \lambda Q\lambda g.\left[Q\lambda y.love'(y)(gy)\right][\lambda P.\forall z[man'(z) \rightarrow Pz]]\)
   \item \(= \lambda g[\lambda P.\forall z[man'(z) \rightarrow Pz](\lambda y.love'(y)(gy))]\)
   \item \(= \lambda g[\forall z[man'(z) \rightarrow \lambda y.love'(y)(gy)](z)]\)
   \item \(= \lambda g[\forall z[man'(z) \rightarrow love'(z)(gz)]]\)
   \end{enumerate}

f. \textit{his dog} \(\leadsto \lambda v\iota y[dog'(y) \land poss(v, y)]\)

g. \textbf{His dog loves every man} \(\leadsto \lambda g[\forall z[man'(z) \rightarrow love'(z)(gz)]][\lambda v\iota y[dog'(y) \land poss(v, y)]]\)
   \begin{enumerate}
   \item \(= \forall z[man'(z) \rightarrow love'(z)(\lambda v\iota y[dog'(y) \land poss(v, y)])(z)]\)
   \item \(= \forall z[man'(z) \rightarrow love'(z)(\iota y[dog'(y) \land poss(z, y)])]\)
   \end{enumerate}

- \textit{Happy?}
   
   Well, it’s not like saying that you can’t move over a coindexed pronoun is any more satisfying.
7 Principle B

- Jacobson gets Principle B by assuming that:

1. Object pronouns belong to a special syntactic subcategory: \( NP_{\ [+p]} \).

2. By default, the lexical category of a verb has its arguments specified as \([-p]\). Consequently, no verb can combine with an object pronoun even after the application of pronominal-Geaching.\(^1\)

3. There is a combinator which I’ll call \( \mathcal{R} \) which:
   a. Syntactically, maps a category \((S/NP)/NP\) to \((S/NP)/NP_{\ [+p]}\).
   b. Semantically, “irreflexivizes” the relation:
      If \( f \) is the characteristic function of a relation \( Q = \{ (x, y) \mid xPy \} \), then \( \mathcal{R}Q \) is the subrelation \( \{ (x, y) \mid xPy \wedge x \neq y \} \).
      More precisely, \( \mathcal{R}f \) is not false but rather undefined when its two arguments have the same value.

- So the only sort of verb which might combine with an object pronoun is one denoting an irreflexive relation. Consequently you can’t express reflexive meaning with a clause that has a pronoun in its object.

- Of course, even after \( \mathcal{R} \) applies, either \( G \) or \( Z \) will have to apply, for the verb to swallow the object pronoun:

\[
(16) \quad \text{Obama criticized him.}
\]

a. criticize: \((S/\backslash NP)/NP \ : \ \text{criticize}'\)

b. \( \mathcal{R}[(S/\backslash NP)/NP \ : \ \text{criticize}'] = (S/\backslash NP)/NP_{\ [+p]} \ : \ \lambda y \lambda x x \neq y. \text{criticize}'(y)(x) \)

c. \( G[\mathcal{R}[(S/\backslash NP)^{NP}/NP \ : \ \text{criticize}']] \)
   i. \( = (S/\backslash NP)^{NP}/NP_{\ [+p]}^{NP} \ : \ \lambda g \lambda v [\lambda y \lambda x x \neq y. \text{criticize}'(y)(x)](gv) \)
   ii. \( = (S/\backslash NP)^{NP}/NP_{\ [+p]}^{NP} \ : \ \lambda g \lambda v \lambda x x \neq v. \text{criticize}'(gv)(x) \)

d. \( G(\mathcal{R}(\text{criticize})) \) him
   \( \mapsto \ [(S/\backslash NP)^{NP}/NP_{\ [+p]}^{NP} \ : \ \lambda g \lambda v \lambda x x \neq v. \text{criticize}'(gv)(x)] \ [NP_{\ [+p]}^{NP} \ : \ \lambda z. z] \)

\(^1\)Jacobson is not this explicit about the feature clash. But something like this is required: an NP must be \([-p]\) by default.
i. $= [(S\backslash NP)^{NP}: \lambda v \lambda x_{x \neq v}.\text{criticize}'(\lambda z.(v))(x)]$

ii. $= [(S\backslash NP)^{NP}: \lambda v \lambda x_{x \neq v}.\text{criticize}'(v)(x)]$

e. $\text{Obama} \mapsto S/(S\backslash NP) : \lambda P.Po$

f. $G(\text{Obama}) \mapsto G[S/(S\backslash NP) : \lambda P.Po]$

i. $= [S^{NP}/(S\backslash NP)^{NP} : \lambda R \lambda w.[\lambda P.Po](Rw)]$

ii. $= [S^{NP}/(S\backslash NP)^{NP} : \lambda R \lambda w.Rwo]$

g. $\text{Obama} \text{ criticizes him} \mapsto G(\text{Obama}) (G(\mathcal{R}(\text{criticize}))(\text{him}))$

i. $= [S^{NP}/(S\backslash NP)^{NP} : \lambda R \lambda w.Rwo][(S\backslash NP)^{NP} : \lambda v \lambda x_{x \neq v}.\text{criticize}'(v)(x)]$

ii. $= [S^{NP} : \lambda w.[\lambda v \lambda x_{x \neq v}.\text{criticize}'(v)(x)](w)(o)]$

iii. $= [S^{NP} : \lambda w_{w \neq o}.[\lambda v \lambda x_{x \neq v}.\text{criticize}'(w)(o)]$

iv. $= \text{‘the set of people who Obama criticized, excluding Obama’}$

- Now recall that ‘binding’ by a quantifier requires application of $Z$, which identifies the subject argument with the argument of the object pronoun.

In case the object pronoun is simple—hence, denotes the identity function—$Z$ reflexivizes the function.

Obviously enough, reflexivizing an irreflexive function—$Z\mathcal{R}f$—won’t have a very happy result. Thus (17) is ruled out as nonsense. More precisely, it violates the irreflexive presupposition associated with $\mathcal{R}(\text{love})$.

(17) * [Every man]$_k$ loves him$_k$.

- Jacobson describes in detail the problems which the standard semantic account has with Principle B.

1. It can’t rule out a reflexive interpretation of the Principle B compliant expression “Obama$_3$ criticized him$_8$,” since 3 and 8 might happen to be the same individual.

To rule this out, the standard account has to posit, it seems, that $g(n) \neq g(m)$ for any $n$ and $m$.

But then (??) would entail that Bill thinks that no one other than himself is living in his house, which seems wrong.

(18) Bill$_k$ thinks that at the moment no one is living in his$_k$ house.
See the paper for the persistence of the problem

What’s important here is that the problem does not arise for Jacobson, who does not need to restrict any assignment functions.

2. Another problem is that Principle B fails to rule out certain bad valuations of variables,

(19) * Every candidate \(k\) thinks that he\(_k\) should say that he\(_k\) will praise him\(_k\).

This valuation will in fact be permitted by Principle B at LF, because it is the interpretation will get for this LF:

```
S
  /\   
S  /
  S
  /
  S
  /
  VP
  /
  S
  /
  S
  /
  says
  /
  S
  /
  he\(_8\) praise him\(_7\)
```

The way out of this for the standard account involves a fair number of complications.

8 Functional questions and answers

Verbs are geached into functions over pronominal functions. So a ‘functional question’ really is a request to identify a certain function. and the answer—a pronominal phrase—really does denote a function, not an entity or whatever.

9 Extraction

As it happens, Jacobson’s system is incompatible with a Steedman-style account of extraction.

(20) Every man wonders who his mother loves

Steedman would require that “his mother” composes with “loves.” But this will not be possible for Jacobson without massive revision of the entire system.