Free and bound variables in natural language

Simon Charlow (simon.charlow@rutgers.edu)

October 19, 2015

1 Semantics for pronouns

1.1 Some basic data

- A pronoun is **free** when its value is fixed by the context in which the pronoun is uttered:

(1) Simon likes her paper.

(2) John walked in. He sat.

- A pronoun is **bound** when its value is fixed not by some feature of the context, but in some sense *sentence-internally* (notice these sentences in principle allow either bound or free construals of the pronouns):

(3) No man$_i$ marries his$_i$ mother.

(4) John met the linguist$_i$ who knows her$_i$ mother.

- Sometimes pronouns must either be bound or free. This is a plausibly syntactic fact which falls under the rubric of the so-called Binding Theory (NB: the sense of *Binding* is related to, but ultimately distinct from our usage of *bound*).

(5) No linguist$_i$ should hate his$_{i/*j}$ own work.

(6) No linguist$_i$ likes him$_{*i/j}$.

1.2 Assignment functions

- The standard semantics for pronouns largely follows how variables are interpreted in formal languages like predicate logic and the lambda calculus (as well as certain programming languages).
- Like FOPL and the semantics of the λ-calculus, it relies on **assignment functions**.
- An assignment function is a device that tells us how **variable** elements like pronouns get interpreted. You can think of assignment functions as features of the context in which an utterance is made.
- More formally, an assignment function is (for our purposes) a function from natural numbers ($\mathbb{N} = 1, 2, 3, \dots$) to individuals. Assignment functions can be total or partial. Here is a simple partial assignment function.

$$\begin{bmatrix} 1 \to \text{Uni} \\ 2 \to \text{Fluffy} \\ 3 \to \text{Porky} \end{bmatrix}$$

- We think of interpretation as **relative to** assignment functions:

$$[\![\textsf{she}_n]\!]^g \coloneqq g\,n$$

- So for example interpreting $\textsf{she}_2$ relative to an assignment that maps 2 to Fluffy gives... Fluffy!

$$[\![\textsf{she}_2]\!]^{\begin{bmatrix}1\to\text{Uni}\\2\to\text{Fluffy}\\3\to\text{Porky}\end{bmatrix}} = \begin{bmatrix}1\to\text{Uni}\\2\to\text{Fluffy}\\3\to\text{Porky}\end{bmatrix} 2 = \text{Fluffy}$$

1.3 Assignments and the grammar

- How to fold assignment-dependent meanings into the grammar we've built? One simple way is to generalize to the worst case, i.e. suppose that *everything's* denotation is relative to an assignment function, though only some things really *depend* on the assignment function for their meaning.

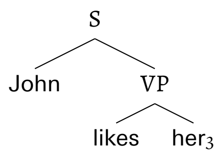
$$[\![\textsf{John}]\!]^g = \textsc{j} \qquad [\![\textsf{likes}]\!]^g = \lambda y.\,\lambda x.\,\textsc{likes}(x,y)$$

- If all interpretations are given relative to assignment functions, our rules for composing meanings must reflect this. This just involves adding a superscripted g to all of our invocations of $[\![\cdot]\!]$ in **FA** and **PM**:

  Functional application
  If A is a branching node with a daughter B of type β and a daughter C of type $\langle\beta,\alpha\rangle$, then for any g, $[\![A]\!]^g \coloneqq [\![C]\!]^g\,[\![B]\!]^g$.

  Predicate modification
  If A is a branching node whose daughters B and C are both of type $\langle e,t\rangle$, then for any g, $[\![A]\!]^g \coloneqq \lambda x.\,[\![B]\!]^g\,x \wedge [\![C]\!]^g\,x$.

- Given these definitions, assignment dependence bubbles up the tree (rather like presupposition, come to think of it). If something behaves like a variable (i.e. has a denotation that can vary depending on the choice of assignment function), so will nodes dominating it:



$$\begin{aligned}[\![\ \cdot\ ]\!]^g &= [\![\text{VP}]\!]^g\,[\![\textsf{John}]\!]^g\\ &= [\![\textsf{likes}]\!]^g\,[\![\textsf{her}_3]\!]^g\,[\![\textsf{John}]\!]^g\\ &= (\lambda y.\,\lambda x.\,\textsc{likes}(x,y))\,(g\,3)\,\textsc{j}\\ &= \textsc{likes}(\textsc{j},g\,3)\end{aligned}$$

- Here, the pronoun causes both VP and S to depend on the assignment function in a way they would not if *her* was replaced with, say, Sue.

1.4 Another view of assignment-dependence

- Equivalently, we may think of interpretations as functions from assignment functions into values. This lets us give a less syncategorematic treatment of pronouns:

$$[\![\textsf{she}_n]\!] \coloneqq \lambda g.g\,n$$

- Doing things this way requires us to recognize a new type, the type of assignment functions. Call it a. Then any type τ in the the old semantics is replaced with $\langle a, \tau\rangle$ in a semantics that thinks of meanings as functions from assignments into "normal" values.
- The recipes for FA and PM on this perspective are essentially the same as before, but we turn the superscripted g's into real arguments:

$$\textsf{FA} : \lambda g.\, [\![C]\!]\, g\, ([\![B]\!]\, g)$$
$$\textsf{PM} : \lambda g.\, \lambda x.\, [\![B]\!]\, g\, x \wedge [\![C]\!]\, g\, x$$

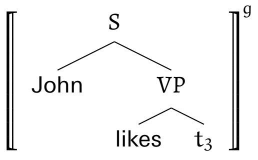
- This involves more parentheses, and folks would really prefer to forget about assignment functions, so we tend to adopt the superscript view. H&K jump through some hoops to get us to ignore the fact that assignment functions are always present, all the time, even when things don't depend on them. But don't be fooled. Assignment-dependence pervades the grammar.

1.5 Traces

- Important assumption: extraction gaps, aka **traces**, have a pronominal semantics (i.e. they are variables). That means they get a subscript and rely on the assignment function for their meaning.

$$[\![t_n]\!]^g \coloneqq g\, n$$

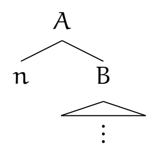
- The meaning of John likes t_3 is the same as the meaning of John likes her$_3$.



$= \textsc{likes}(\textsc{j}, g\, 3)$

2 Abstraction

- We have a way to give a semantics to the trace—i.e. in terms of a variable. We now need a way to *bind* this variable.
- As a first step, we'll allow structures like the following (this corresponds to the implementation H&K adopt in later chapters, though not Chapter 5). The index n is an **abstraction operator**:



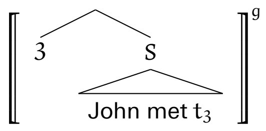
- We give a special, syncategorematic rule for predicate abstraction:

  Predicate abstraction
  If A is a branching node with daughters $n \in \mathbb{N}$ and B of type τ (for some τ), then A has type $\langle e, \tau\rangle$ and for any g, $[\![A]\!]^g \coloneqq \lambda x.[\![B]\!]^{g[n \to x]}$

- Relies on the notion of re-writing or re-mapping a value:

$$\begin{bmatrix} 1 \to \text{Uni} \\ 2 \to \text{Fluffy} \\ 3 \to \text{Porky} \end{bmatrix} [3 \to x] = \begin{bmatrix} 1 \to \text{Uni} \\ 2 \to \text{Fluffy} \\ 3 \to x \end{bmatrix}$$

- So our key interpretive principles number three (though **PM** is more or less dispensable): **FA**, **PM**, and **PA**.
- Let's try it out with an example:



$$\begin{aligned} &= \lambda x.\, [\![\text{S}]\!]^{g[3 \to x]} && \text{By } \textbf{PA} \\ &= \lambda x.\, \textsc{met}(\textsc{j}, g[3 \to x]\, 3) && \text{See above} \\ &= \lambda x.\, \textsc{met}(\textsc{j}, x) && = \end{aligned}$$

- The result is a property, the characteristic function of the set of individuals x such that John met x. It's almost as if the trace had never been there!
- Notice that the result no longer mentions the assignment function. Abstraction turns an assignment-dependent meaning into an assignment-independent meaning. **Binding** is achieved.
- Notice that if you go bottom-up with a specific assignment g in mind (say, the one that maps 3 to Porky, as above), you can end up doing a lot of work that eventually gets tossed out. For that reason, if you care about assignments, you should interpret things *top-down*.
- Notice also that while we'll usually be abstracting out of sentences, **PA** doesn't require that. It so happens that **PA** as stated yields a function *from individuals* to something else, but we could imagine generalizing this (though we'd need some variables other than pronouns!).

3 Relative clauses

3.1 Basic case

- We now have everything we need to derive a basic case like the woman who John met. See Figure 1.
- The key bits: the gap introduces a variable. A co-indexed abstraction operator uses that variable to derive a property. And that property is intersected with the head noun, exactly as in the cases of modification we saw last week.
- Here, we assume the relative pronoun is vacuous, i.e. has as its meaning the identity function over properties. We could also give build modification into its semantics if we desired.

3.2 Binding pronouns

- Given that pronouns *also* denote variables, we predict that an abstraction operator at the edge of a relative clause can bind both. This is borne out in DPs like the man who praised himself. See Figure 2.
- So long as the trace and pronoun bear the same index, the bound interpretation results. If the pronoun bears a different index from the trace, a free interpretation results (which happens to be ungrammatical here).

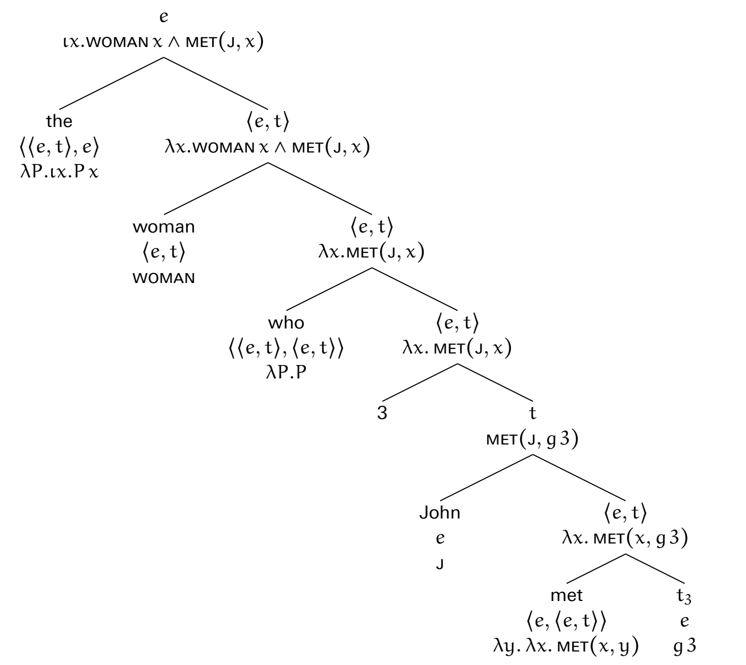


Figure 1: $[\![$the woman who John met$]\!]^g$ (assuming defined-ness, given any g)

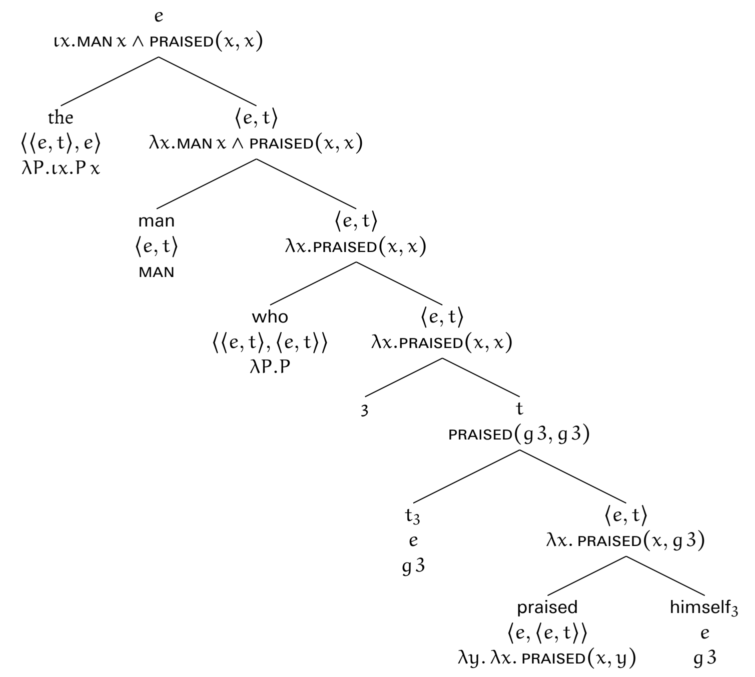
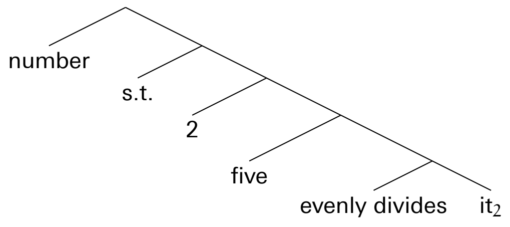


Figure 2: $[\![$the man who praised himself$]\!]^g$ (assuming defined-ness, given any g)

3.3 *Such-that* relatives

- *Such-that* relatives lack a gap, but they can be assigned a semantics along parallel lines.

(7) The smallest number such that five evenly divides it



- This is something of a coup for our view that gaps have the same semantics as pronouns. Since the semantics of gaps is by assumption the same as the semantics of pronouns, *such-that* relatives can be handled using the same tools (despite the fact that they lack a gap).

4 Wrapping up

4.1 Over-generation concerns

- Enriching our semantics to fix under-generation opens us up to over-generation.
- Possibility of **vacuous abstraction**. Given an assignment g, the following either denotes the unique man (if Sue met $g(2)$), or nothing (if there is no unique man, or if Sue did not meet $g(2)$).

(8) The man who [1 [Sue met t_2]]

- It's worth running through a calculation to convince yourself of this.
- We can also imagine configurations where the abstraction operator binds a pronoun but **not a gap**!. Notice that that t_2 could be bound by some other higher operator, leading to truly strange readings.

(9) The man who [1 [she$_1$ met t_2]]

- More generally, traces **never** have free uses like pronouns. They only ever have bound uses. This is not accounted for by our simple-minded semantics and syntax.
- Finally, **crossover** configurations like the following are predicted to be possible, contrary to fact (again, I'll suggest you try to verify this with a calculation):

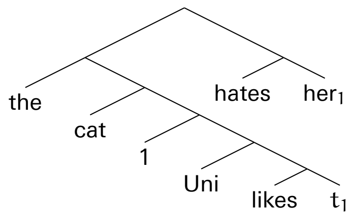
(10) *The man who [1 [he$_1$ likes t_1]]

(11) *?The man who [1 [his$_1$ mother likes t_1]]

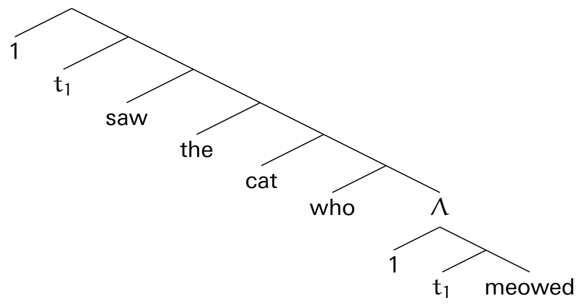
- How to fix? Need, it seems, properly representational constraints—i.e. constraints which tell you that certain trees are not well-formed. That is, we hope that the syntax can clean up our mess here. See H&K for some discussion of candidate constraints on trees.

4.2 Free versus bound

- Two facts. First, an abstraction index in the syntax needs to **c-command** the trace or pronoun for the pronoun to be bound. Consider the following structure.



- The trace t_1 is bound by the abstraction operator 1, but the VP-internal pronoun he_1 is not.
- Second, when an abstraction index c-commands an identical index, the latter takes precedence, binding any co-indexed variables in its scope. Consider a RC like *who saw the cat who meowed*:



- Here, Λ denotes a constant function over assignments, i.e. the property of meowing. That is, t_1 is bound **inside** Λ. Moral: a variable's free-ness bubbles up the spine of the tree until it meets a co-indexed abstraction index.
- In sum:

  An abstraction index n binds a pronoun x_n iff n c-commands x_n, and there is no lower occurrence of an abstraction index n that c-commands x_n.

- Do either of these things remind of you of something? Both of these are precisely how the lambda calculus's treatment of variable binding works. In other words, our treatment of pronominal/trace binding in natural language relies on a close analogy with the lambda calculus's treatment of variable binding.