564 Lecture 19 Oct. 28, 1999

1 Relative clauses

Relative clauses describe a property of their head nouns -- they're just like any other modifier in an NP. The idea is that a relative clause is a one-place predicate that, just like any other predicate that we've got, denotes the characteristic function of a set. So, in the examples in (1), the clause should be of type <e,t> and denote all the things that have the particular property in question:

1. (a) [which I bought] = c.f. of the things such that I bought them (e.g. iBook, etc.)

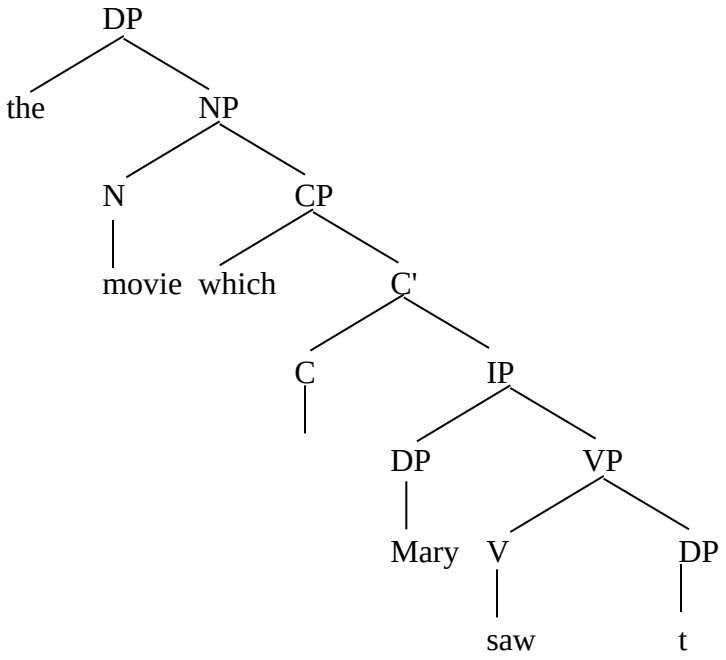
(b) [which I spent talking to Sue] = c.f. of the things such that I spent them talking to Sue (e.g., days, hours, etc.)

(c) [whose book appalled me] = c.f. of the things such that their book appalled me (e.g. some author)

(d) Macy Gray is [who I like].

The next question is, what's the type of the trace? it should be something like <e>, because that would mean that it could combine suitably with the verbs as their argument. Let's just consider the tree structure H&K give (their #3) and ask, as they do, what the antecedent of the trace could be:

2.



Now, in order for the trace to have the type <e>, its antecedent would have to be of type <e>, too, in order for it to pick up the right type. But there is no possible antecedent of the right type available in the structure. If its antecedent was "movie", then it'd have to have type <e,t>, which isn't the right type to combine with "saw" (by any of our rules, even PM). If its antecedent was the whole DP, (the movies which...), it'd get the right type -- but you'd get into a problem of infinite regress: the movie which Mary saw [the movie which Mary saw [the movie which Mary saw...]]] etc.

Rather, what we need to give the value of the trace is a *variable*, whose referent is determined by a particular *assignment*, just like we did with quantifiers in predicate calculus. That is, we basically need to say that the interpretation of the trace is dependent upon other factors, which we're about to determine. Remember in predicate calculus that we decided that pronouns were variables, and only got their reference via a particular assignment function?
So, we had an example like in (2) (from Lecture 7):

3. a. Invite (x,y)

b. Ann likes Bill, so she invited him to the party.
c. Carol felt sorry for Dave, so she invited him to the party.

In 3b and c, the clause "she invited him to the party" means two different things. In b, it means that Ann invited Bill to the party. In c it means that Carol invited Dave to the party. We took care of this by saying that the assignment function was different in each context, and hence assigned different individuals to the variables x and y in each context:

4. **Model:**

⟦ c ⟧ = Carol
⟦ d ⟧ = Dave
⟦ a ⟧ = Ann
⟦ b ⟧ = Bill
⟦ s ⟧ = Sue
⟦ j ⟧ = John
⟦ Invite ⟧ = {<x,y> __DxD | Invite(x,y) }
Assignment functions
g(x) = Ann
g(y) = Bill
h(x) = Carol
h(y) = Dave
i(x) = Sue
i(y) = John

We're going to use the same treatment for the trace in our relative clause. It's going to be a variable, and variables only get a denotation under a particular assignment.

Now, H&K go into a huge inductive pit trying to get you to figure out we need more than one variable in order to treat sentences with more than one pronoun, or a pronoun and a trace, like 5:

5. John said that the book which he wrote t is appalling.

Now, if there's just one variable, and let's say the value of the variable under a given assignment is the individual John, the clause "he wrote t" will turn out to mean "John wrote John". This is not what we want.

In predicate calculus, we used x and y to be different variables (as in "Invite(x,y)" above), but in linguistic theories, people use the item itself plus a numerical subscript: we're going to treat the numerical subscript as the variable. So the sentence in 5 is usually tagged as follows:

6. $John_1$'s an author. The book that he_1 wrote t_2 is appalling.

This will ensure that the variable represented by "he" in the sentence is a different variable from that represented by "t" in the sentence.

Let's just briefly consider our original example of some assignment functions from predicate calculus. In a particular context, we had an assignment function g and we defined it as follows:

7. g(x) = Ann
   g(y) = Bill

That is, it's a function that took the variable x as an argument and spit out the value "Ann", and took the variable y as an argument and spit out the value "Bill". That is, the table representing the denotation of the assignment function g would look like this:

8.

$$g: \begin{pmatrix} x \rightarrow \text{Ann} \\ y \rightarrow \text{Bill} \end{pmatrix}$$

Now, we're not using "x" and "y" for our variables, but rather the set of natural numbers. So let's re-treat our predicate logic sentence above using natural numbers:

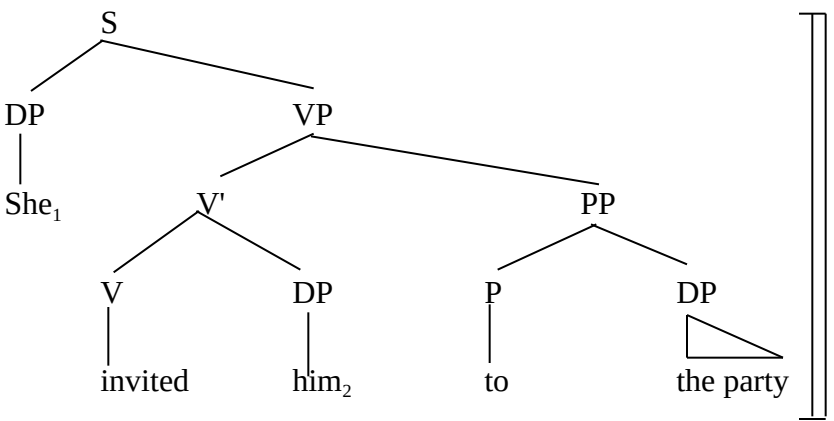
9. Ann_1 likes $Bill_2$, so she_1 invited him_2 to the party.

In this context, the assignment function g can take the value "1" as its argument and in that case it spits out the value "Ann", or it can take the value "2" as its argument and spit out the value "Bill". So we can represent our assignment function's operation in either of the following ways. We're going to call our assignment function "a" from now on, instead of "g", because H&K do:

10. a. a(1) = Ann
a(2) = Bill

b. a: $\begin{pmatrix} 1 \rightarrow \text{Ann} \\ 2 \rightarrow \text{Bill} \end{pmatrix}$

Ok. So, now how do we incorporate our assignment function into our semantics? It's part of the interpretation function, so we're going to write it (just the way we did in predicate calculus) as a superscript on our interpretation function double braces. All interpretations will now be relative to a particular assignment. So the interpretation of "She invited him to the party" will look like this:

11. $[\![$ 

$]\!]^{\begin{pmatrix} 1 \rightarrow \text{Ann} \\ 2 \rightarrow \text{Bill} \end{pmatrix}}$

(We could just as easily have simply written "a" up in the right-hand corner, which stands for the function that's represented in the table.) So, to interpret this structure, we'll start at the top, and work our way down. Let's see what happens

when we try to interpret the subject DP: we can get as far as this with the rules that we have:

12. $\left[\!\!\left[\begin{array}{c} \text{DP} \\ | \\ \text{She} \end{array}\right]\!\!\right]^a \quad = \quad [[\text{She}_1]]^a$ (by N.N.)

That's fine - but what is the denotation of $[[\text{She}_1]]^a$? Our rule for terminal nodes won't cover it. In order to interpret these pronouns (and traces), we're going to have to have a special rule for them that tells the semantics to look at the assignment function to find out what they are. Here's the rule, H&K call it the "Pronouns and Traces" rule (p. 116):

13. *Pronouns and Traces Rule.*

If α_i is a pronoun or a trace, a is a variable assignment, and i is a number in the domain of a, then $[[\alpha_i]]^a = a(i)$.

So, now when we get to something like $[[\text{She}_1]]^a$ in our interpretation, we'll know what to do: we check out what the assignment function says about it.

(Note: we've now got 5 interpretation rules: Non-Branching Nodes, Functional Application, Predicate Modification, Terminal Nodes I: Lexical Entries and Terminal Nodes II, The Sequel: Pronouns and Traces. We have to revise all these rules so that the assignment-independent part of their interpretation stays the same when the interpretation function is made relative to an assignment. The way to do this is to make a definition telling us what the interpretation of a given tree is under a particular assignment when it doesn't make any difference what that assignment is, and then make all our rules apply to nodes under a particular assignment. Using the definition, then, we'll be able to figure out the denotation of assignment-independent things under the assignment, and the assignment itself will tell us the interpretation of assignment dependent things. The definition H&K give is the following (#9 p. 94):

14. For any tree α, α is in the domain of [[ ]] iff for all assignments a and b, $[[\alpha]]^a = [[\alpha]]^b$.

If α is in the domain of [[ ]], then for all assignments a, $[[\alpha]] = [[\alpha]]^a$.

So now we'll be chugging along doing our interpretation under a particular assignment (because that's the way we do all our interpretations), and we might get to a terminal node like **laugh** that needs to be interpreted. Our terminal node rule for lexical terminals says that [[**laugh**]] is specified in the lexicon, and so it is: [λx D . x laughs]. But we need to know, not [[**laugh**]], but rather $[[\textbf{laugh}]]^a$, so we check out our definition. Our definition says (ignoring the first part, which basically tells us to check if $[[\textbf{laugh}]]^a$ is the same as $[[\textbf{laugh}]]^b$ for any possible assignment a and b -- we can intuitively see that the meaning of **laugh** will satisfy this requirement) ... as I was saying, our definition says, $[[\textbf{laugh}]]^a = [[\textbf{laugh}]]$. And luckily we know what [[**laugh**]] is, so we know what $[[\textbf{laugh}]]^a$ is and we can therefore interpret our terminal node **laugh** with respect to assignment a. Then, all the rest of our rules we'll just rewrite so that all their interpretations are w/r to an assignment, and we'll be done. For completeness' sake , I'll write all of our interpretation rules so far right here, and include a new one, the sixth, which I'm about to introduce:

L.T. *Lexical Terminals*

If α is a terminal node occupied by a lexical item, then [[α]] is specified in the lexicon. (and then our definition in 14 will tell us what $[[\alpha]]^a$ is for the specific casee).

N.N. *Non-Branching Nodes*

If α is a non-branching node and β is its daughter, then for any assignment a, $[[\alpha]]^a = [[\beta]]^a$.

F. A. *Functional Application*

If α is a branching node and {β,γ} the set of its daughters, then for any assignment a, if $[[\beta]]^a$ is a function whose domain contains $[[\gamma]]^a$, then $[[\alpha]]^a = [[\beta]]^a ([[\gamma]]^a)$.

P. M. *Predicate Modification*
If α is a branching node and {β,γ} the set of its daughters, then, for any assignment a, if $[[\beta]]^a$ and $[[\gamma]]^\alpha$ are both functions of type <e,t>, then
$[[\alpha]]^a = [\lambda x \quad D_e \,.\, .[[\beta]]^a(x) = .[[\gamma]]^a(x) = 1]$

P.T. *Pronouns and Traces Rule.*
If α_i is a pronoun or a trace, a is a variable assignment, and i is a number in the domain of a, then $[[\alpha_i]]^a = a(i)$.

(to be introduced below:)
P.A. *Predicate Abstraction*
If α is a branching node whose daughters are β_i and γ, where β is a relative pronoun or "such", and i |N, then for any variable assignment a, $[[\alpha]] = [\lambda x \quad D_e \,.\, [[\gamma]]^{a^{x/i}}]$.

2 Traces

Ok, so now we've treated pronouns. What about traces? Well, essentially, we're going to say that traces *are* pronouns — at least, A-bar traces are. They represent a place in the syntax where a variable is. First, I submit for your consideration the following three sentences:

15. (a) Wilma is a woman who Betty likes.
(b) Wilma is a woman such that Betty likes her.
(c) Wilma is a woman who Betty likes her.
((d) Wilma is a woman who Betty sometimes wonders if she likes Fred.)

Our semantics is going to assign a-c all the exact same meaning; as far as our semantics goes, in fact, these sentences will be indistinguishable. (d) is just included to demonstrate that while (c) is pretty odd, there are places where a pronoun instead of a trace sounds almost grammatical, if non-standard. Indeed, in some languages, whereever you get wh-words, rather than a trace, you have a resumptive pronoun.

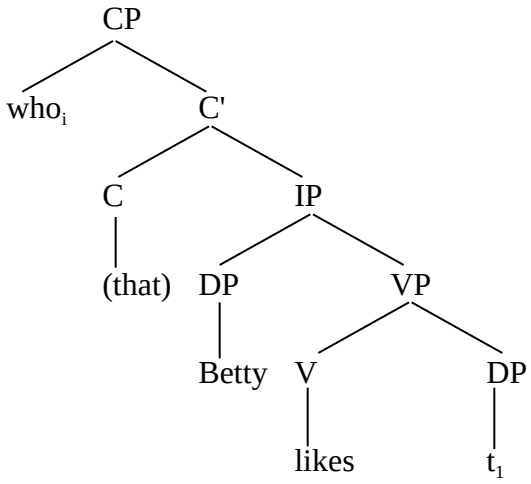
So, the denotation of a trace is just going to be whatever individual a particular assignment assigns to the trace's index. As far as the semantics goes, 16 a and b are equally well-formed, and on a given assignment a, will mean exactly the same thing:

16. (a) Betty likes him_1.
    (b) Betty likes t_1

3 Relative Clauses and Lambda Abstraction

Ok .So that's what traces are. Now, onward. A sentence like 16b is a subtree of a relative clause like 17:

17. (a) who_i Betty likes t_1.
    (b)



Now, what's the denotation of that IP? We've just said that, relative to an assignment a, it'll be the same as a denotation for "Betty likes him", i.e. a truth value. And what's the denotation of the whole CP? Well, we've decided that it's the same as a regular modifier, like "empty" or "gray" -- type <e,t>. So something must be happening on the way from IP to CP to change the denotation of the tree from <t> to <e,t>. And we want it to do so in such a way that an argument given to the function that's the denotation of the resulting modifier is interpreted as

being in the set of things that are values for the variable within IP that make the IP true -- i.e. within the set of things that Betty likes.

The thing that we're going to assume does the work for us (for the moment) is the wh-word, the relative pronoun. We're going to assume that the C head is just vacuous (although we'll see later that the real story is more probably the other way around). We're going to say that the appearance of a wh-word as the sister of some other node triggers an assignment-altering operation on that node, and gives a function that maps things to true iff they're in the set of things that are values for thevariable within the sister that make the sister node true. That sounds more complicated than it is. What it does is the following, called Predicate Abstraction, or sometimes "Lambda Abstraction", our sixth interpretation rule, repeated from above:

18. P.A. *Predicate Abstraction*
If α is a branching node whose daughters are β_i and γ, where β is a relative pronoun or "such", and i |N, then for any variable assignment a,
$[[\alpha]] = [\lambda x \quad D_e . [[\gamma]]^{a^{x/i}}]$.

Now, there's one more thing we need to understand about what we can do with assignments before we can understand this rule. See the supersuperscript x/i that is superscripted to the superscript a that refers to the assignment function we're using? What it's doing is changing the assignment function. It's changing it by saying, whatever this assignment function used to assign to the variable i before, now it's assigning x to it.

Here's some examples of modifying assignments. Let's reconsider our original assignment a:

19. (a) a: $\begin{pmatrix} 1 \rightarrow \text{Ann} \\ 2 \rightarrow \text{Bill} \end{pmatrix}$

(b) $a^{Mary/1}$, which can of course be equivalently written as

a: $\begin{pmatrix} 1 \rightarrow \text{Ann} \\ 2 \rightarrow \text{Bill} \end{pmatrix}^{\text{Mary/1}}$

What this does is it takes whatever a used to assign to the integer 1, and says, replace that with Mary, i.e. it changes what a assigns to the integer 1, if anything (if a didn't assign anything to the integer 1 before, it does after it's modified with a superscript, as indicated). So, the modified function represented in 19b could be written in table form as in (20):

20. a': $\begin{pmatrix} 1 \rightarrow \text{Mary} \\ 2 \rightarrow \text{Bill} \end{pmatrix}$

And, if you want to go really bananas, you can modify the modified assignment function, as in (a), which ultimately gets you the modified function a''' represented in table form in 21(b):

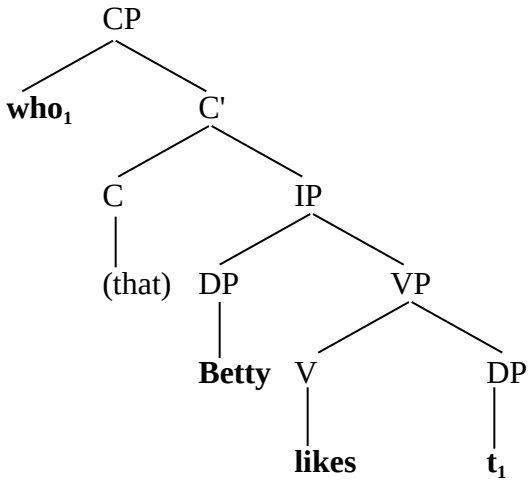
21. (a) $[[a^{\text{Mary/1}}]^{\text{Ann/1}}]^{\text{Betty/3}}]$.

(b) a''': $\begin{pmatrix} 1 \rightarrow \text{Ann} \\ 2 \rightarrow \text{Bill} \\ 3 \rightarrow \text{Betty} \end{pmatrix}$

Ok. So what's our predicate abstraction rule doing? It's interpreting branching nodes which have relative pronouns with some index i as their daughter. What it does is say, this node is a function from entities (x) to truth values. It gives the value "true" if the interpretation of its other daughter node (the one that's not a relative pronoun) is true when x is the interpretation of the index i. So what it does is represent the characteristic function of the set of things that make the embedded sentence true when they're inserted in place of the variable -- i.e. it's the c.f. of the set of entities that Betty likes, in our example relative clause from 17.

Let's work through the interpretation of that relative clause as an example.

22. (a)



Now, the CP node matches the environment for our sixth rule, P.A., so

(b) $[[CP]] = [\lambda x \quad D_e \,.\, [[C']]^{x/1}\ ]$

P.A. applied to CP

(c) $=[\lambda x \quad D_e \,.\, [[IP]]^{x/1}\ ]$

N.N. (and vacuity of C) applied to C'

(d) $=[\lambda x \quad D_e \,.\, [[VP]]^{x/1}\ ([[DP]]^{x/1})\ ]$

F.A. applied to IP

(e) $=[\lambda x \quad D_e \,.\, [[VP]]^{x/1}\ ([[\mathbf{Betty}]]^{x/1})\ ]$

N.N. applied to DP

(f) $=[\lambda x \quad D_e \,.\, [[V]]^{x/1}\ ([[DP]]^{x/1})([[\mathbf{Betty}]]^{x/1})\ ]$

F.A. applied to VP

(g) $=[\lambda x \quad D_e \,.\, [[\mathbf{likes}]]^{x/1}\ ([[\mathbf{t_1}]]^{x/1})([[\mathbf{Betty}]]^{x/1})\ ]$

N.N. applied to V and DP

(h) $=[\lambda x \quad D_e \,.\, [[\mathbf{likes}]]^{x/1}\ ([[\mathbf{t_1}]]^{x/1})(\text{Betty})\ ]$

L.T. (plus def. 14) applied to **Betty**

(i) $=[\lambda x \quad D_e \,.\, [[\mathbf{likes}]]^{x/1}\ (x)(\text{Betty})\ ]$

P.T. applied to $\mathbf{t_1}$

(j) $=[\lambda x \quad D_e \,.\, [\lambda y \quad D_e \,.\, [\lambda z \quad D_e \,.\, z \text{ likes } y]](x)(\text{Betty})\ ]$

L.T. (plus def. 14) applied to **likes**

(k) $=[\lambda x \quad D_e \,.\, [\lambda z \quad D_e \,.\, z \text{ likes } x](\text{Betty})\ ]$

Definition of lambda notation.

(l) $=[\lambda x \quad D_e \textbf{ .} \text{ Betty likes x}]$

Definition of lambda notation.

Homework

1. In predicate calculus, "x" stood for a variable, and got a denotation under an assignment. In our notation now, the indices on traces and pronouns stands for a variable, and they are what gets a denotation under an assignment. "x" is a possible value that can be assigned to the variable. In our predicate calculus, there was a different letter that we used in the same sort of function that we now use "x" for in the Predicate Abstraction rule. What was that letter, and what's the sort of function? (Hint: it's in lecture 7 or 8 (not both!))

2. Practice with revising assignment functions: Here's an assignment function a:

a: $\begin{pmatrix} 2 \rightarrow \text{Betty} \\ 4 \rightarrow \text{Fred} \\ 6 \rightarrow \text{Wilma} \\ 8 \rightarrow \text{Barney} \end{pmatrix}$

Draw the tables that represent the following functions which are revisions of a:

(a) $a' = a^{\text{Betty/3}}$

(b) $a'' = a^{\text{Dino/1, Barney/2}}$

(c) $a''' = a^{\text{BamBam/4, Pebbles/6, Betty/8}}$

(d) $a'''' = a^{\text{BamBam/1, Fred/2, Wilma/3, Barney/4, Pebbles/5, Dino/7}}$

3. (a) Draw a tree for the sentence "Dino is the brown dinosaur that licked Fred" (Use IPs instead of Ss)

(b) Prove that the sentence in (a) is true iff Dino is the unique brown dinosaur that licked Fred. -- i.e., derive the meaning of the whole IP by applying our rules to find out the meanings of subtrees, given appropriate definitions for the lexical items that we haven't defined so far. *On the way you will have to invent a new meaning for the copula "is"*, which so far we have assumed to be vacuous, i.e. meaningless. Use the format for proving meanings that I used in

showing the meaning of the relative clause in 22 above -- that is, you can just refer to the interpretation of a node, assuming that the substructure of the node is what is represented in the tree you drew in 3a above.