

Verbs!
LING 553
September 15, 2008

1. HOW TO COMBINE THINGS

1.1. “Entity” + “Action” = Truth value

Last class, it was suggested that we have:

$$(R1) \left[\begin{array}{c} \overline{\text{NP}} \\ | \\ \alpha \end{array} \right] = \llbracket \alpha \rrbracket \quad (R2) \left[\begin{array}{c} \overline{\text{VP}} \\ | \\ \alpha \end{array} \right] = \llbracket \alpha \rrbracket \quad (R3) \left[\begin{array}{c} \overline{\text{V}} \\ | \\ \alpha \end{array} \right] = \llbracket \alpha \rrbracket \quad (R4) \left[\begin{array}{c} \text{S} \\ / \quad \backslash \\ \alpha \quad \beta \end{array} \right] = \llbracket \alpha \rrbracket \dots \llbracket \beta \rrbracket$$

and that in particular,

- $\llbracket \text{Julie} \rrbracket = \text{Julie}$ (an entity, e)
- $\llbracket \text{snores} \rrbracket = \text{an act of snoring}$ (an action, a)
- (R4): $\llbracket \text{S} \rrbracket = \text{TRUE}$ if and only if the entity does the action
 = TRUE if and only if $\llbracket \text{NP} \rrbracket$ engages in $\llbracket \text{VP} \rrbracket$
 (or rather: if and only if $\llbracket \alpha \rrbracket$ engages in $\llbracket \beta \rrbracket$)

It’s a reasonable starting point—but we’ll reject it, for a few reasons:

- Not every VP denotes an action you can engage in.¹ (R4) will work fine if VP is **snores** or **sleeps**, but what about **knows French** or **likes Charles** or **is Canadian**, which are things you can’t exactly “engage” in?
- (R4) will assign a meaning to an S with daughters α , β ; but wouldn’t it be nice if the rules didn’t have to refer to that kind of syntactic detail?

A digression...

Right now, (R1)-(R3) also refer to syntactic detail: there’s a rule for NP, a rule for VP, and a rule for V. By preference, we’d like a way to avoid that. Fortunately, those rules are similar enough that we can do so:

If X is a node with one daughter α , then $\llbracket X \rrbracket = \llbracket \alpha \rrbracket$.

We’ll call this the **Non-Branching Node Rule**, or NBNR.²

¹ Quick reminder: don’t confuse linguistic objects and their interpretations! A verb phrase like **snores** denotes an action; $\llbracket \text{snores} \rrbracket$ is an action; but it’s wrong to say that **snores** is an action. (It’s not. It’s a word.)

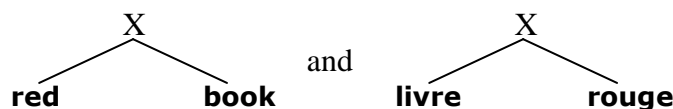
² Note the shorthand of using simply “X” rather than drawing the structure. For simplicity, we’ll generally put in just the top node, with the understanding that it means “along with all the structure under it”.

- If we take out the syntactic detail, things will go wrong if we try to interpret, say, a branching node whose daughters are **red** and **book**: neither one is an action that the other engages in. And they'll go even worse for a VP like **sees Charles**, where there *is* an action and an entity, but we don't want to say that the entity engages in the action in this case.

Another digression...

Someone asked, quite reasonably, if that's really an issue: after all, in **Julie snores**, it's an entity and then an action, but in **sees Charles**, it's the other way around.

But in fact, when we interpret structures, all we need is the geometry—in fact, the linear order of the daughters doesn't matter, and is really a question of syntax (or phonology?). **Julie snores** should have the same interpretation as **snores Julie** in a language where verbs precede subjects; for that matter, in an SOV language, we'd have **Charles sees** as the VP, but Charles still doesn't do the seeing. Similarly at other points: since $[[\text{red}]] = [[\text{rouge}]]$ and $[[\text{book}]] = [[\text{livre}]]$, we ought to get the same meaning out of both



even though the order of the daughters is different.

1.2. A unified rule

Instead, following Frege's concept of sentences as "saturated" thoughts and verb phrases as "unsaturated" thoughts, we can say the following: a verb phrase is something that, if you give it an entity, it'll give you a truth value. But what that is is simply a function: specifically, it's

- (1) that function from entities to truth values such that, given an entity, it returns TRUE if and only if that entity snores

Now we need a way to combine them. Well, we've got an entity, and we've got a function that takes entities. A logical way to combine them is:

the meaning of S is the result of applying the function $[[VP]]$ to the entity $[[NP]]$

and we can write that in a non-node-specific way:

if X has two daughters α and β , where $[[\beta]]$ is a function, $[[X]] = [[\beta]]([[\alpha]])$.

(Note: Yanyan asks about $[[\text{snows}]]$, which doesn't seem to be a function that takes the subject of the sentence as an argument. There's not much to say to that—she's pretty much right. It

looks like **snows** and other weather verbs aren't going to denote functions; they're going to denote truth values, i.e., they denote predicates that are already saturated. That, to me, doesn't seem to be a problem: there won't be a one-to-one correspondence between parts of speech and kinds of meaning. We'll revisit that fact as we proceed.)

2. SEMANTIC TYPES

Lydia suggests that there should be a way of connecting the meaning in (1) for **snores** with the meaning in (2), for **sleeps**, and perhaps even the meaning in (3), for **would hate the trees I'm drawing**:

- (2) that function from entities to truth values such that, given an entity, it returns TRUE if and only if that entity sleeps
- (3) that function from entities to truth values such that, given an entity, it returns TRUE if and only if that entity would hate the trees I'm drawing

What they have in common is that they're all "unsaturated predicates"—they're all the same kind of function, namely functions from individuals to truth values.

So let's lay out a system of "semantic types" that we can use to tie all this together. We already have truth values, t , which were collected into a set D_t , and entities, e , which are members of D_e . Building from them, the semantic types are as follows:

- e and t are semantic types;
- if σ, τ are semantic types, then $\langle \sigma, \tau \rangle$ is a semantic type;
- nothing else is a semantic type.

So for instance, $\langle e, t \rangle$ is a semantic type—in particular, it's the type of functions from individuals to truth values (i.e., $f: D_e \rightarrow D_t$). Other types include:

- $\langle e, e \rangle$
- $\langle e, \langle e, t \rangle \rangle$
- $\langle \langle e, t \rangle, e \rangle$
- $\langle \langle e, t \rangle, \langle \langle e, t \rangle, t \rangle \rangle$
- $\langle e, \langle e, \langle e, \langle e, e \rangle \rangle \rangle \rangle$

We won't use all of these types, but they're there.

Generally, things of type $\langle \sigma, \tau \rangle$ are functions from things of type σ to things of type τ ; and in general, D_τ is the set of things of type τ . Thus we can write that $\llbracket \text{snores} \rrbracket$ is a function of type $\langle e, t \rangle$; or that $\llbracket \text{snores} \rrbracket \in D_{\langle e, t \rangle}$.

A final note about notation: occasionally $\langle e, t \rangle$ is abbreviated et ; that's not much help on its own, but it certainly can be easier to read $\langle et, \langle et, t \rangle \rangle$ than $\langle \langle e, t \rangle, \langle \langle e, t \rangle, t \rangle \rangle$. Very occasionally, $\langle e, \langle e, t \rangle \rangle$ is abbreviated eet . H&K don't use these, and in general I won't either, in part because it's important to remember that $\langle e, \langle e, t \rangle \rangle \neq \langle \langle e, e \rangle, t \rangle$, and brackets are more perspicuous. But you may see them elsewhere, so now you know.

2.1. A revision to the unified rule

Earlier we had

If X has two daughters α and β , where $\llbracket \beta \rrbracket$ is a function, $\llbracket X \rrbracket = \llbracket \beta \rrbracket(\llbracket \alpha \rrbracket)$.

But that's not a very good rule—if the daughters of X are **sleeps** and **snores**, then it's true that one of them is a function, but $\llbracket X \rrbracket \neq \llbracket \text{sleeps} \rrbracket(\llbracket \text{snores} \rrbracket)$, or vice versa. Fortunately, we can now restrict things based on semantic types by saying:

If X has two daughters α and β , where $\llbracket \beta \rrbracket$ is type $\langle \sigma, \tau \rangle$ and $\llbracket \alpha \rrbracket$ is type σ , then $\llbracket X \rrbracket$ is type τ , and $\llbracket X \rrbracket = \llbracket \beta \rrbracket(\llbracket \alpha \rrbracket)$

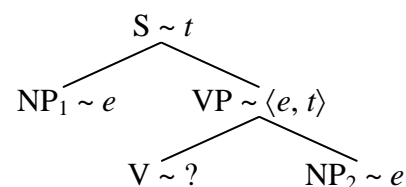
We'll call this the **Rule of Function Application** (FA), and it'll underlie everything we do. And for completeness, we should add:

The Lexical Meaning Rule

If α is a terminal node, its interpretation $\llbracket \alpha \rrbracket$ is its lexical meaning.

3. TRANSITIVE VERBS

FA is entirely neutral with respect to syntactic nodes. This means our semantics can be *type-driven*: we can combine things not based on, say, syntactic node labels, but based their semantic types. At each point in the derivation, we can see what the semantic type is, and therefore how the pieces combine.



So having done intransitive verbs: what is the denotation of a transitive verb like **sees** in **Julie sees Charles**? $\llbracket \text{sees Charles} \rrbracket$ should have type $\langle e, t \rangle$. If $\llbracket \text{Charles} \rrbracket$ has type e , and we want to combine the pieces via function application, we have two choices:

- Make $\llbracket \text{sees} \rrbracket$ the kind of thing that $\llbracket \text{Charles} \rrbracket$ takes as an argument. But of course, things of type e aren't functions; they don't take arguments. That leaves...
- Make $\llbracket \text{sees} \rrbracket$ the kind of thing that takes $\llbracket \text{Charles} \rrbracket$ as an argument. So:
 - it needs to be a function that takes an e ... $\langle e, \dots \rangle$
 - ...and returns an $\langle e, t \rangle$ object. $\dots, \langle e, t \rangle$
 That is: it needs to be an $\langle e, \langle e, t \rangle \rangle$.

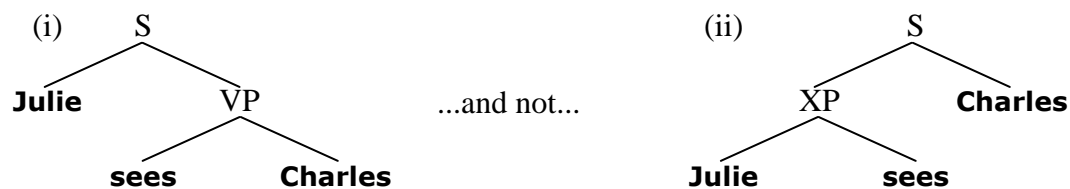
What function is it? Specifically:

[[sees]] = that function from entities to [functions from entities to truth values] such that, given an entity (call it *a*), it returns that function from entities to truth values such that, given an entity (call it *b*), it returns TRUE if and only if entity *b* sees entity *a*.

Formatted a little more perspicuously:

[[sees]] = that function from entities to [functions from entities to truth values] such that:
given: an entity (call it *a*),
it returns: that function from entities to truth values such that:
given: an entity (call it *b*),
it returns: TRUE if and only if entity *b* sees entity *a*.

Note: why “*b* sees *a*” and not “*a* sees *b*”? Because syntacticians tell us that the structure is:



...and we have no reason to doubt them. More on this next time.