564 Lecture 12 Oct. 5, 1999

1. Back to sqaure 1

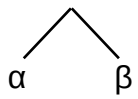
So, we've gotten fairly sophisticated at predicate calculus and understanding how to translate certain kinds of English expressions into predicate calculus. What we don't have, however, is an explicit, rule-by-rule procedure for translation, that makes clear the relationship between the syntactic structure of the sentence and its meaning.

First, some notes about syntax:

1. The syntax creates a structure that is interpreted by the semantics. In a sense, its job is to put together meaningful elements in a way that the semantic component can deal with and compute a total meaning from. It might have its own little internal rules (if you've taken syntax you'll have seen that it certainly does), but its main goal is to produce an interpretable structure.

2. Most syntacticians these days assume *binary branching*. That is, the syntax takes two items, and sticks them together, to create a third item.
(a)

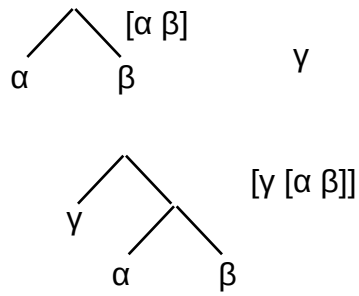
α β



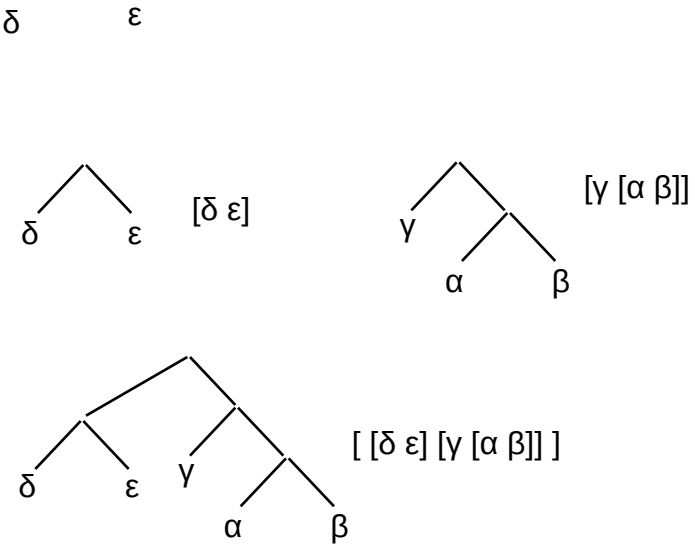
[α β]

Then it can stick another item onto that.

(b)



Then it might stick two more items together and then stick the result of that operation onto the result of the preceding two sticking-together operations.
(c)



(Let's say "Merge" instead of "sticking together").

One way, and the simplest way, to make explicit the relationship between the syntax and the semantics is to assert that all composition in the syntax (all merging) is putting together a function and some appropriate item that it can take as its argument. This is called *functional application* and it satisfies the *Principle of Compositionality* in a strict and transparent way:

3. *Principle of Compositionality*: We compute the informational content of a complex expression from the informational content of its parts.

Functional Application: all merging (sisterhood) puts together a function and its argument. The denotation of the whole is the value of the function for that argument.

Now, how do we institute this idea that every pair of sisters is a function and its argument? So far, we've been saying that the denotation of, say, an intransitive verb is a set of entities in the world. In predicate logic, the idea was that we look at the set of entitites, and decide whether or not the given argument of that predicate was in the set or not. If it was, we said that the predicate-argument combination was true, if it wasn't, we say it's false.

We can implement the idea of functional application by saying that that intransitive verb has a denotation that takes entities as its domain and maps them onto a range of truth values, 1 and 0. In a sense, the "decision" process has been included in the meaning of the verb itself.

4. **Function**: a special kind of ordered pair, or *relation*. For any first element in an ordered pair, there is one and only one second element. The set of all possible first elements for a given function is called its *domain* , and the set of all possible second elements is its *range*.

2 Computing the meaning of a syntactic structure from its parts

For example, let's take the intransitive sentence "Ann smokes". Before, we said the denotation of "smokes" was {x | x smokes}, and that the sentence was true iff [[Ann]] {x | x smokes}. Now, we're going to say that the denotation of "smokes" is a function from entities to truth-values — i.e., it's a set of ordered pairs, the first of which is an individual and the second of which is a truth value. (Note that before we used "U" to indicate the universe of discourse, i.e. the set of all possible entities. H&K use "D", and from now on, we will too. Also, H&K use

boldface instead of quotes to indicate strings of the object language; we'll try to do that mostly but no guarantees).

5. *Notation*:
$f : \alpha \rightarrow \beta$
for every x α, f(x) β

6. [[**smokes**]] $\Leftrightarrow$

a) *Before*:
$\Leftrightarrow \{x \mid x \text{ smokes}\}$ (predicate notation for sets)
or, in *list* form, in a world where Ann, John and Sue smoke:
$\Leftrightarrow$ {Ann, John, Sue}

b) *Now*:
$\Leftrightarrow f: D \rightarrow \{0, 1\}$
For every x D, f(x) =1 iff x smokes
(condition notation for functions)

or, in *list* form, in a world where Ann, John and Sue smoke, and Bill and Joe don't, and that's all the entities there are:
$\Leftrightarrow$ f:={ <Ann, 1>, <John, 1>, <Sue, 1>, <Bill, 0>, <Joe, 0>}

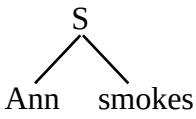
or, in *table* form:

f :=
Ann -->	1
John -->	1
Sue -->	1
Bill -->	0
Joe -->	0

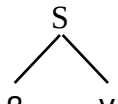
So, let's say that the above function is the denotation of the lexical item **smokes**, and the denotation of proper names is the one we've assigned them, that is, individuals. [[**Ann**]] = Ann, [[**John**]] = John, and so on.

Now, if our syntactic tree for "Ann smokes" were just 7, we'd be done:

7.



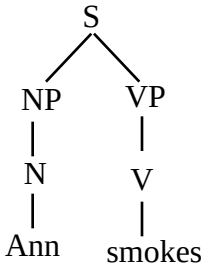
We'd just make explicit that the meaning of the whole is the combination of the meaning of its parts: the function denoted by **smokes** takes the individual denoted by **Ann** as its argument, and produces a truth value (in this particular example, 1) as the denotation of the combination of its two parts (i.e. the S node). That is, we'd write the following rule:

8. If α has the form 

, then [[α]] = [[γ]] ([[β]])

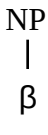
Of course, that's not the syntactic tree for "Ann smokes". It's more complicated than that. Really, the structure of "Ann smokes" is better represented by 8 (perhaps; cf. Bare Phrase Structure, Chomsky 1995):

8.

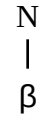


In order to have a fully specified mapping from the syntax to the semantics, we need to have rules telling us, for every node in the syntactic tree, what its denotation is. The meaning of N is made up of the meaning of its parts, i.e. [[Ann]], the meaning of NP is made up of the meaning of its parts, i.e. [[N]],

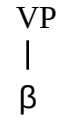
and similarly for the VP node and its dependents. Then we can say that the meaning of the S node is made up of the meaning of its parts, i.e. the meanings of [[NP]] and [[VP]] (once we know them). So we need a set of rules specifying all that:.

9. If α has the form 

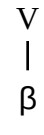
, then [[α]] = [[β]]

10. If α has the form 

, then [[α]] = [[β]]

11. If α has the form 

, then [[α]] = [[β]]

12. If α has the form 

, then [[α]] = [[β]]

That, in combination with the rule in 8, will enable us to prove the denotation of the sentence in 8 to be 1 in the world under consideration. You simply take each node in the tree, and substitute an appropriate meaning for it, using the rules for structure in 8-12 and the denotations of the lexical items in (6b).

13. So, our semantics includes:

A: a bunch of possible types of denotations

i) elements of D, the set of individuals
ii) elements of {1,0}, the set of truth values
iii) functions from D → {1,0}

Note: already we've gotten away from first-order predicate logic, which didn't contain any functions (except the interpretation function, which we've still got). As we'll see below, the functions we have so far are all characteristic functions of sets of individuals, so we're not really truly away from first-order yet -- but we've got the tools necessary to take us there.

B. a bunch of lexical items, with denotations like those listed in A above:

[[**Ann**]] = Ann
[[**Joe**]] = Joe
...etc. for other proper names

[[**smokes**]] f: D → {1,0}
For all x D, f(x) = 1 iff x smokes

[[**snores**]] f: D → {1,0}
For all x D, f(x) = 1 iff x snores
... etc. for other intransitive verbs

C. a bunch of rules (8-12) that give the meaning of a piece of syntactic structure, deriving that meaning from the subparts of the structure by functional application.

4 Back to sense and reference

Ok, so above we've said that the meaning of "Ann smokes" is 1 in a world where Ann smokes. But really, what we know when we know the meaning of the sentence "Ann smokes" is not *whether* its true or not, but the *conditions* that it takes to make it true. That is, our idea of the meaning of *smokes* is really the most like the "condition" notation for functions given above, and not like the table or list notation for functions. Again, that 's the diff. between sense and reference.

5. Characteristic functions for sets:

Now, we haven't really gotten away from first-order logic yet, because the denotation for intransitive verbs that we've considered so far is simply the *characteristic function* for the set that we used as the old denotation. The characteristic function of a set just maps members of that set to truth value 1, and all other elements of D to 0. And we can get back to the set from the function by saying that the characterized set of a function f is {x | f(x) =1}.

14. a) *Characteristic function* f of a set A:

f: D → {0,1}

for every x A, f(x) = 1

b) The set A *characterized* by a function f:

{x | f(x)=1}

So what we've got so far is a way of making first-order logic compositional, with proper names for terms and functions for predicates.

Just to be sure we're clear on our notation, we'll go over H&K's illustration of the difference between assuming functions as denotations of predicates and assuming sets as denotations of predicates.

15. *if sets are the denotations of predicates*:

(a) [[**sleep**]] = {Ann, Jan, Sue}
(b) [[**snore**]] = {Ann, Sue}
(c) Ann [[**sleep**]]
(d) [[**snore**]] [[**sleep**]]
(e) | [[**sleep**]] ∩ [[**snore**]] | = 2

(remember |A| = cardinality of A)

16. *if functions are the denotations of predicates*:

(a) [[**sleep**]] = $\begin{pmatrix} \text{Ann} \rightarrow 1 \\ \text{Jan} \rightarrow 1 \\ \text{Sue} \rightarrow 1 \\ \text{Maria} \rightarrow 0 \end{pmatrix}$

(b) [[**snore**]] = $\begin{pmatrix} \text{Ann} \rightarrow 1 \\ \text{Jan} \rightarrow 0 \\ \text{Sue} \rightarrow 1 \\ \text{Maria} \rightarrow 0 \end{pmatrix}$

(c) ******Ann [[**sleep**]]

(c') <Ann, 1> [[**sleep**]]

(c'') [[**sleep**]](Ann) = 1

(d) ******[[**snore**]] [[**sleep**]]

(because [[**snore**]] contains one element, <Jan, 0>, that [[**sleep**]] does not).

(d') {x : [[**snore**]](x) = 1} {x : [[**sleep**]](x) = 1}

(e) ****** | [[**sleep**]] ∩ [[**snore**]] | = 2

(because [[**snore**]] now has 3 elements in common with [[**sleep**]]: <Ann, 1> <Sue, 1> and <Maria, 0>).

(e') |{x : [[**snore**]](x) = 1} ∩ {x : [[**sleep**]](x) = 1}| = 2

Homework: exercise on sentence connectives, p. 23. What we're looking for is a) a new type of denotation, which is exemplified by the connectives (hint: it's a function) b) a lexical entry for the connectives specifying their truth conditions, and c) a rule mapping the syntax of the given structures onto the meaning for those structures.