Lab Session 4: Schoenfinkelizing and Writing Functions for Transitive Verbs
September 30, 2013

Goal: Why, at the end of last discussion (9/26), did we decide to write the denotation for *loves* in the way that we did? The goal of this lab session is to understand and motivate denotations like (1) a bit more.

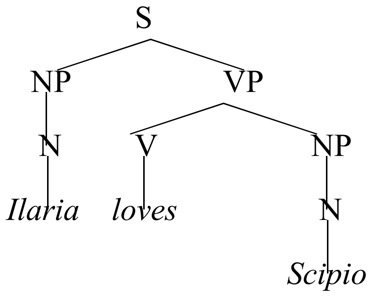
(1) [[*loves*]] = g: $D_e \rightarrow D_{<e,t>}$, for all y ∈ D_e, g(y) =
Read: the function g such that it maps entities to type <et> functions, where for all entities y, application of g to y yields….

$h_y : D_e \rightarrow D_t$, for all x ∈ D_e, $h_y(x)$ = 1 iff x loves y
Read: ...a function h_y such that it maps entities onto truth values, where for all entities x, application of h_y to x yields true if and only if x loves y.

1. What syntax we assume

- Syntactic structure is important to semantics because it determines how different words can combine together.

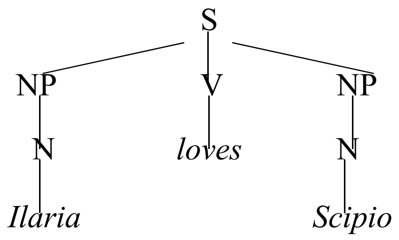
- We assumed a syntactic structure like (2) for the sentence *Ilaria loves Scipio*.

(2)


- In (2), the transitive verb V *loves* combines with an NP (its direct object; *Scipio*) to form a VP. The VP then combines with the subject NP (*Ilaria*) to form a sentence S.

- We will assume **binary branching** structures like in (2) for this class. Syntax has given us a number of reasons to think that the structure should be as in (2) and not as in (3).

  - **Key insight from syntax:** The subject and direct object of a verb do not stand in a symmetrical relationship with each other.

(3)
No!


2. A naïve analysis of the meanings of transitive verbs

- In class on Thursday, we assumed the binary branching structure in (2) and, as such, decided that we would treat transitive verbs as functions of type <e,<e,t>>.

- We can describe a function of type <e,t> as being a **one-place** function because it is a function that combines with one thing (here, an entity).

  - We can also describe a function of type <e,<e,t>> as being **1-place**. Why? Because it is a function that combines with one thing (here, an entity).

- Let's forget for a moment that we want to assume a binary branching structure like in (3). Then, let's rewind two weeks to our discussion of **relations**. What did we say about relations?

  - **Ordered tuples** (e.g., pairs, triples, 4-tuples, 5-tuples…) can help us talk about *relations* (or, statements) that hold between two or more individuals.

  - A relation like *is bigger than* can be defined as an ordered pair.

Let's say that D_e = {Leopold, Dmitri, Sebastian}

Sebastian is the biggest goat, Dmitri is the middle sized goat, Leopold is the smallest goat

How can we write the relation *is bigger than* as a set of ordered pairs? (let's call that relation R_{bigger})

(4) R_{bigger}=

Now, I ask you to write the characteristic function for the set in (4). We'll call that function f_{bigger}. The characteristic function takes an ordered pair and returns 1 iff that ordered pair is in the set in (4).

Let's write it in table notation, since that's a clear way to write characteristic functions.

(5) $$f_{bigger} = \left[\begin{array}{l} \langle \text{Sebastian, Dmitri} \rangle \rightarrow 1 \\ \langle \text{Sebastian, Leopold} \rangle \rightarrow 1 \\ \langle \text{Sebastian, Sebastian} \rangle \rightarrow 0 \\ \\ \langle \text{Dmitri, Dmitri} \rangle \rightarrow 0 \\ \langle \text{Dmitri, Leopold} \rangle \rightarrow 1 \\ \langle \text{Dmitri, Sebastian} \rangle \rightarrow 0 \\ \\ \langle \text{Leopold, Dmitri} \rangle \rightarrow 0 \\ \langle \text{Leopold, Leopold} \rangle \rightarrow 0 \\ \langle \text{Leopold, Sebastian} \rangle \rightarrow 0 \end{array} \right]$$

Question 1: If f_{bigger} were the denotation of *is bigger than*, then we can say this is a how-many-place function (I.e., is it 1-place? Is it 2-place?)? Why?

Question 2: If f_{bigger} were the denotation of *is bigger than*, would the syntax look like? Would it look more like (2) or like (3)? Why?

Programmatic decision: Based on what we know about syntax, we'll always assume binary branching structures.

> …a consequence of this decision is that transitive verbs will always be analyzed as 1-place functions. Even though a transitive verb has two syntactic arguments (a direct object and a subject), we will feed these arguments in one at a time.

3. Schoenfinkelization!

- Thanks to a fellow named Moses Schoenfinkel, we have a way to take an *n*-place function (like the characteristic function that we wrote above for *is bigger than*, f_{bigger} in (5)) and turn it into a series of 1-place functions so that we can keep our binary branching structures.

- There's two ways to Schoenfinkelize a 2-place function like f_{bigger}.

Way One to Schoenfinkelize: Left-to-right

Let's define a new function, g_{bigger}, which is the **left-to-right** Schoenfinkelized version of f_{bigger}. The function g_{bigger} applies to the **first** (i.e., lefthand) member of each ordered pair in (5). The output of g_{bigger} is a function that applies to the **second** (i.e., righthand) member of each ordered pair in (5) and returns 1 iff the first member *is bigger than* the second member.

(6) $$g_{bigger} = \left[\begin{array}{ll} \text{Sebastian} \rightarrow & \left[\begin{array}{l} \text{Dmitri} \rightarrow 1 \\ \text{Leopold} \rightarrow 1 \\ \text{Sebastian} \rightarrow 0 \end{array}\right] \\ \text{Dmitri} \rightarrow & \left[\begin{array}{l} \text{Dmitri} \rightarrow 0 \\ \text{Leopold} \rightarrow 1 \\ \text{Sebastian} \rightarrow 0 \end{array}\right] \\ \text{Leopold} \rightarrow & \left[\begin{array}{l} \text{Dmitri} \rightarrow 0 \\ \text{Leopold} \rightarrow 0 \\ \text{Sebastian} \rightarrow 0 \end{array}\right] \end{array}\right]$$

⇑ this is the input to g_{bigger} (AKA, the argument of g_{bigger})

⇑ this function is the output of g_{bigger} (AKA, the value that each argument is mapped onto by g_{bigger})

When the function g_{bigger} is applied to Sebastian, then it outputs a type <e,t> function which maps any goat onto 1 iff Sebastian is bigger than that goat.

What is produced when the function g_{bigger} is applied to Dmitri?

What is produced when the function g_{bigger} is applied to Leopold?

Way Two to Schoefinkelize: Right-to-left

Let's define a new function, h_{bigger}, which is the **right-to-left** Schoenfinkelized version of f_{bigger}. The function h_{bigger} applies to the **second** (i.e., righthand) member of each ordered pair in (5). The output of is h_{bigger} is a function that applies to the **first** (i.e., lefthand) member of each ordered pair in (5) and returns 1 iff the second member *is smaller than* the first member.

(7) $$h_{bigger} = \left[\begin{array}{ll} \text{Dmitri} \rightarrow & \left[\begin{array}{l} \text{Sebastian} \rightarrow 1 \\ \text{Dmitri} \rightarrow 0 \\ \text{Leopold} \rightarrow 0 \end{array} \right] \\ \text{Leopold} \rightarrow & \left[\begin{array}{l} \text{Sebastian} \rightarrow 1 \\ \text{Dmitri} \rightarrow 1 \\ \text{Leopold} \rightarrow 0 \end{array} \right] \\ \text{Sebastian} \rightarrow & \left[\begin{array}{l} \text{Sebastian} \rightarrow 0 \\ \text{Dmitri} \rightarrow 0 \\ \text{Leopold} \rightarrow 0 \end{array} \right] \end{array} \right]$$

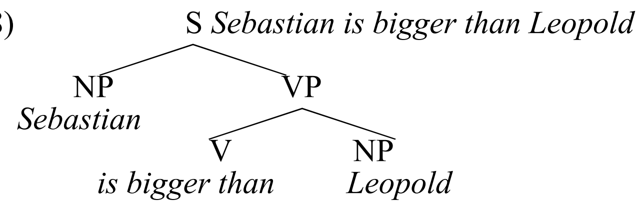
!!! Why is right-to-left Schoenfinkelization very useful to think about, given the way in which we wrote our ordered pairs in (4) and (5)?

The way we wrote our ordered pairs in (4) (which was the relation *is bigger than*) and in (5) (which was the not-Schoenfinkelized characteristic function of *is bigger than*) was like this: **<subject, object>.** As Heim and Kratzer say (1998: 31), this is a **totally arbitrary** choice that we probably have made because subjects come before objects in English.

Given the way that we have defined h_{bigger} in (7), h_{bigger} is combining first with the object of the verb *is bigger than*. The output of h_{bigger} is three new, type <e,t> expressions: *is bigger than Dmitri*, *is bigger than Leopold*, and *is bigger than Sebastian*. These functions then go on to combine with the subject.

Upshot of all this: By Schoenfinkeling in this way, we have defined a function h_{bigger} that seems to capture the way that a transitive verb like *is bigger than* combines with its arguments in the syntax.

(8)



We can write out the denotation for *is bigger than* that will get us the right result given the structure in (8).

(9) [[*is bigger than*]] = $h_{\text{bigger}} : D_e \rightarrow D_{<e,,t>}$, for all $y \in D_e$, $h_{\text{bigger}}(y) = h_{\text{bigger-than-y}} : D_e \rightarrow D_t$,
for all $x \in D_e$, $h_{\text{bigger-than-y}}(x) = 1$ iff x is bigger than y

The entire table diagram (enclosed by the biggest set of brackets) is a way of representing h_{bigger}.

Question: Looking back at our table diagram for h_{bigger} in (7), which parts of the table correspond to the function $h_{\text{bigger-than-y}}$ (AKA, the second line of the denotation in (9))?

4. In-class exercise

Let's say that D_e = {Basil, Sybil, Polly}
The following are true statements in the actual world: Basil loves Sybil, Polly loves Basil, and Sybil loves Polly.

a. Please write the relation *loves* as a set of ordered pairs.

b. Please write the two-placed (AKA, not-yet-Schoenfinkelized) characteristic function (call it f_{loves}) for the relation you wrote above. Please write it in table notation.

c. Please Schoenfinkelize your function f_{loves} <u>right-to-left.</u> Call the new function you make h_{loves}.

d. Using (9) as a model, please write a denotation for *loves* based on your answer for (c).

e. Put everything together. Write out the full derivation of the truth value for the sentence *Basil loves Sybil*.

(i) Draw a tree for the structure
(ii) Write out lexical entries for *Basil*, *Sybil*, and *loves*
(iii) Do subproofs for both NPs and VP.
(iv) Put the subject NP and the VP together to make S and get a truth value.

Since all expressions will be evaluated in the actual world (w_0), don't worry about writing your superscript ^{w0} on your denotations.