Introduction to Semantic Theory
Sentences and composition

Dr. Sarah Zobel

Class: May 18, 2016

Connecting back to the previous lecture

The central result of the previous lectures was the introduction of set notation and functional notation for extensions.

- $[\![Peter]\!]^w = \text{Peter}$
- $[\![tree]\!]^w = \{x : x \text{ is a tree in } w\} = \lambda x.\ \text{tree'}(x)(w)$
- $[\![blue]\!]^w = \{x : x \text{ is blue in } w\} = \lambda x.\ \text{blue'}(x)(w)$
- $[\![snore]\!]^w = \{x : x \text{ is snoring in } w\} = \lambda x.\ \text{snore'}(x)(w)$
- $[\![like]\!]^w = \{\langle x, y\rangle : x \text{ likes } y \text{ in } w\} = \lambda y.\lambda x.\text{like'}(y)(x)(w)$
- $[\![give]\!]^w = \{\langle x, y, z\rangle : x \text{ is giving } y \text{ to } z \text{ in } w\}$
  $= \lambda z.\lambda y.\lambda x.\text{give'}(z)(y)(x)(w)$

Aim for today

The aim for today: to extend the picture towards sentences, introduce the basic idea behind modelling composition, and modify the proposal for the extension of verbs

- ⇒ sentences can also be assigned an extension; the combinatoric system will, however, derive a different (but related) aspect of the meaning of sentences, their truth conditions.
- ⇒ We will see: to model composition, we utilize the possibilities given by the function-based model of extensions that we assumed.
- ⇒ We will formulate: two derivational rules (NN) and (FA), which allow us to derive the truth conditions of sentences made up by verbs and proper names.

What about sentences?

To build a model for semantic composition, we not only need to have an assumption about the "starting point" (i.e., the extensions of lexical items), but also about the "end point".

Since the model should capture the semantic ability of speakers: take a closer look at what speakers know and are able to determine about sentences.

Semantic ability of competent speakers

- Competent speakers of a language can understand infinitely many sentences, and also sentences that they have never seen or heard before.

  ⇒ Consequence of the combinatoric nature of meaning

- Competent speakers can determine whether one sentence entails another sentence, or not.

  ⇒ Remember lab class 1

- Competent speakers are able to detect ambiguities.

  ⇒ We have seen and practiced that in the last two lectures.

- Competent speakers know what has to be the case in the world so that a given sentence describes it correctly. They know **the truth conditions of a sentence**.

Truth conditions – I

The truth conditions of a sentence have been famously linked to its meaning by Ludwig Wittgenstein.

> To understand a sentence means to know what is the case if it is true.
> (Wittgenstein 1922)

The methodological problem with this idea is, however, that to make clear what the truth conditions of a sentence are, we need to use language. But if we use language to talk about language, we need to distinguish two levels: **object language** and **meta language**.

Truth conditions – II

To illustrate the necessity for the object vs. meta language distinction, consider the truth conditions for '*Peter is snoring*' in (1).

(1) Peter is snoring is true in a world if and only if Peter is snoring in that world.

Truth conditions – II

To illustrate the necessity for the object vs. meta language distinction, consider the truth conditions for '*Peter is snoring*' in (1).

(1) Peter is snoring is true in a world if and only if Peter is snoring in that world.

In the course of the truth conditions, we change from object language to meta language.

The expression/sentence that we want to describe (the object of our investigations) is given in object language. The language we use to describe the object of our investigations is the meta language.

Truth conditions – III

Wittgenstein's connection between truth conditions and meaning also states that to grasp the meaning of a sentence, we do not need to know whether it is true, or not! This is important to realize.

⇒ The truth conditions of a sentence will be the "end point", i.e., what the formal model should ultimately derive.

But are the truth conditions of a sentence its extension?

The extension of sentences

Since the extension of an expression is supposed to be a "thing"/a set of "things" in the world, the truth conditions cannot be the extension of a sentence.

What would be a good candidate for the extension of a sentence?

The standard assumption, which we will adopt, is that the extension of a sentence is its truth value (true/false) in the world.

Truth-conditions and truth value of a sentence

Connection between the truth conditions of a sentence and its truth value in a world:

Given the truth conditions—if we know what the world is like, we can determine whether the sentence is true (= 1) or false (= 0) in the world.

(2) *Peter is happy*

⇒ *Peter is happy* is true in *w* iff Peter is happy in *w*

World_1: ☺

$[\![\textit{Peter is happy}]\!]^{World_1} = 1$

World_2: ☹

$[\![\textit{Peter is happy}]\!]^{World_2} = 0$

Interim summary

We now have decided on:

- the input of composition: the extensions of the lexical elements in a given sentence
- the output of composition: the truth conditions of the resulting sentence

The next step is to formulate derivational rules that tell us how to combine the extensions of lexical elements to derive the truth conditions of the resulting sentence.

Principle of compositionality

The combinatoric nature of meaning and the creativity of competent speakers suggest hat sentence meaning is built up from the meanings of the lexical expressions that occur in the sentence. This central insight goes back to Gottlob Frege:

(3) Principle of compositionality (Frege's principle)
The meaning of a composite expression is a function of the meaning of its immediate constituents and the way these constituents are syntactically combined.

The goal of the formal model: to derive the truth conditions of a sentence from the extensions of the lexical items contained in it.

Composition as saturation – I

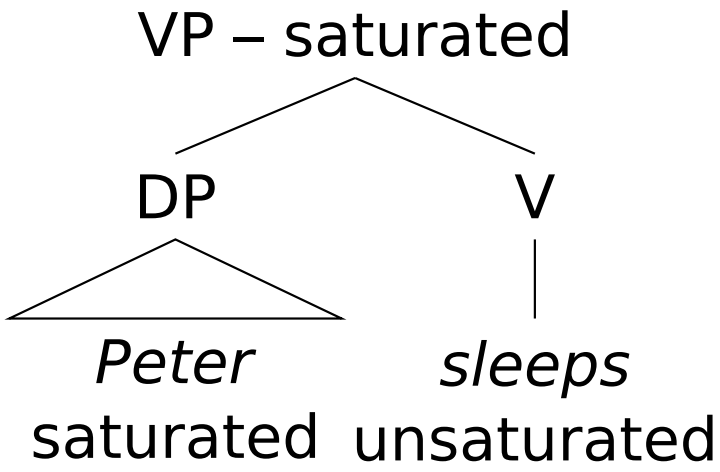
Underlying idea: semantic composition is saturation of unsaturated meaning.

Given the extensions we have determined until now, which classes of lexical items can be said to have unsaturated extensions?

- Saturated extensions: proper names
- Unsaturated extensions: nouns, adjectives, verbs (= functional extensions)

⇒ Sentences also have saturated extensions, since they denote their truth-values.

Composition as saturation – II



- $[\![Peter]\!]^w = \text{Peter}$
- $[\![sleeps]\!]^w = \lambda x.\text{sleep}'(x)(w)$

What is the result of composing all parts of the tree above?
What do we want to derive?

Truth-conditions in formal notation

(4) Formal notation for truth conditions:

$[\![X]\!]^w = 1$ iff Y

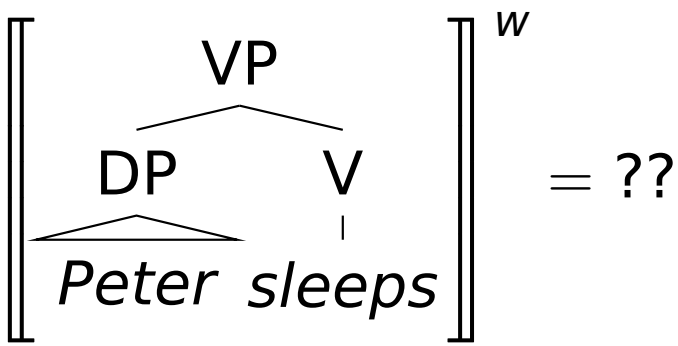
The sentence *X* (in object language!) is true in the world of evaluation *w* if and only if (iff) the circumstances described by Y (in meta language!) obtain in *w*.

⇒ The derivations that are supported by our formal system should result in truth conditions of the form above.

(5) $[\![\textit{Peter sleeps}]\!]^w = 1$ iff sleep'(Peter)(*w*)

A first derivation rule: (NN) – I

Basic assumption: Semantic derivations are guided by the hierarchical structure of LF. Combination of two extensions (i.e., composition) always happens for the daughters of a branching node.



What happens at non-branching nodes in the LF-structure?

A first derivation rule: (NN) – II

At non-branching nodes, nothing happens since there is nothing to combine! Hence, the first derivational rule is:

(6) **Non-branching nodes (NN):**
For a non-branching node α with the single daughter β, $[\![\alpha]\!]^w = [\![\beta]\!]^w$

$$\left[\!\!\left[\begin{array}{c} \text{VP} \\ \text{DP} \quad \text{V} \\ \textit{Peter} \quad \textit{sleeps} \end{array}\right]\!\!\right]^w \overset{2\times(NN)}{=} \left[\!\!\left[\begin{array}{c} \text{VP} \\ \textit{Peter} \quad \textit{sleeps} \end{array}\right]\!\!\right]^w = ??$$

Next step: composition – I

After “getting rid” of non-branching nodes, the lexical expressions are the daughters of a branching node. In this configuration, the extensions of the two expressions are composed. But how?

$$\left[\!\!\left[\begin{array}{c} \text{VP} \\ \hline \textit{Peter} \quad \textit{sleeps} \end{array} \right]\!\!\right]^{w} = [\![\textit{Peter}]\!]^{w} \circ [\![\textit{sleeps}]\!]^{w} = ??$$

Note: The symbol $\circ$ is only a temporary stand-in for composition.

Next step: composition – II

To see what the function notation suggests as a formal counterpart for the idea that composition is saturation, let's substitute the $[\![.]\!]^w$-notation with the corresponding extensions.

$$[\![Peter]\!]^w \circ [\![sleeps]\!]^w = \text{Peter} \circ [\lambda x.\text{sleep'}(x)(w)]$$

What could be a good candidate for a specification of $\circ$?

Remember: the extension of an intransitive verb is the characteristic function of the set of individuals who perform the action denoted by the verb.

Formal counterpart of saturation: function application

The symbol ∘ is best specified as the function provided by *sleeps* applying to the individual provided by *Peter*: we ask for Peter whether he is in the set of people who are sleeping.

$$\left[\!\!\left[\begin{array}{c} \text{VP} \\ \diagup\!\diagdown \\ \textit{Peter}\ \ \textit{sleeps} \end{array} \right]\!\!\right]^{w} \overset{(FA)}{=} [\![\textit{sleeps}]\!]^{w}([\![\textit{Peter}]\!]^{w})$$

$$= [\lambda x.\text{sleep}'(x)(w)](\text{Peter}) = ??$$

This is an application of the derivational rule of function application (FA).

Applying a function in λ-notation

λ-notation allows us to simplify the last step further by applying the so-called λ-conversion rule:

(7) $$\big[\lambda x.\ f[x]\big](y) = f[y]$$

For our example: $[\lambda x.\text{sleep'}(x)(w)](\text{Peter}) \overset{\lambda}{=} \text{sleep'}(\text{Peter})(w)$

Convention: if there is no further composition step to be done to derive the truth conditions of a given sentence, the result of the last λ-conversion is interpreted as the meta-linguistic part of the truth conditions of that sentence.

Hence: $[\lambda x.\text{sleep'}(x)(w)](\text{Peter}) \overset{\lambda}{=}$ **1 iff** $\text{sleep'}(\text{Peter})(w)$

The full derivation: *Peter sleeps*

$$[\![\text{VP} \;[\text{DP } \textit{Peter}]\; [\text{V } \textit{sleeps}]]\!]^w \overset{2\times(NN)}{=} [\![\text{VP} \;[\textit{Peter}]\; [\textit{sleeps}]]\!]^w$$

$$\overset{(FA)}{=} [\![\textit{sleeps}]\!]^w([\![\textit{Peter}]\!]^w)$$

$$= [\lambda x.\text{sleep'}(x)(w)](\text{Peter})$$

$$\overset{\lambda}{=} 1 \text{ iff } \text{sleep'}(\text{Peter})(w)$$

The left part and the last line of the right part give the truth conditions of *Peter sleeps* in formal notation!

A different case: transitive verbs – I

Let's assume that the following VP is the VP of the sentence '*Peter likes Mary*':

$$[\![\text{VP} \;(\text{V}\ \textit{likes},\ \text{DP}\ \textit{Mary})]\!]^w \overset{2\times(NN)}{=\!=} [\![\text{VP}\ \textit{likes}\ \textit{Mary}]\!]^w$$

$$\overset{(FA)}{=\!=}\ ??$$

What is the result of the application of (FA)?

A different case: transitive verbs – II

The functional extension of the verb *likes* is applied to the non-functional extension of *Mary*:

$$[\![\text{VP} \;(\text{V}\ \textit{likes},\ \text{DP}\ \textit{Mary})]\!]^w \overset{2\times(NN)}{=} [\![\text{VP} \;(\textit{likes}\ \textit{Mary})]\!]^w \overset{(FA)}{=} [\![\textit{likes}]\!]^w([\![\textit{Mary}]\!]^w)$$

How can we write down the (FA) rule so that both possible function-argument-configurations found with branching nodes are captured?

Solution: semantic types

Semantic types allow us to state which logical structure the extension of a given expression has: do they denote individuals? characteristic functions of sets of individuals? something different?

- Basic types: e (individuals/entities), t (truth values)
- Functional types:
  for arbitrary types α, β, $\langle \alpha, \beta \rangle$ is also a type

The types α and β that make up a functional type $\langle \alpha, \beta \rangle$ represent the type of the argument α (= input) and the type of the output β of the function. An expression of type $\langle e, t \rangle$ has a functional extension that takes individuals (type e) as input and returns a truth value (type t).

Types of lexical expressions

What are the types of the following expressions?

- $[\![\textit{Peter}]\!]^w = \text{Peter}$
- $[\![\textit{tree}]\!]^w = \lambda x.\ \text{tree'}(x)(w)$
- $[\![\textit{blue}]\!]^w = \lambda x.\ \text{blue'}(x)(w)$
- $[\![\textit{snore}]\!]^w = \lambda x.\ \text{snore'}(x)(w)$
- $[\![\textit{like}]\!]^w = \lambda y.\lambda x.\text{like'}(y)(x)(w)$
- $[\![\textit{give}]\!]^w = \lambda z.\lambda y.\lambda x.\text{give'}(z)(y)(x)(w)$

Types of lexical expressions

What are the types of the following expressions?

- $[\![Peter]\!]^w = \text{Peter}$ e
- $[\![tree]\!]^w = \lambda x.\ \underline{\text{tree'}(x)(w)}$ $\langle e, \underline{t}\rangle$
- $[\![blue]\!]^w = \lambda x.\ \underline{\text{blue'}(x)(w)}$ $\langle e, \underline{t}\rangle$
- $[\![snore]\!]^w = \lambda x.\ \underline{\text{snore'}(x)(w)}$ $\langle e, \underline{t}\rangle$
- $[\![like]\!]^w = \lambda y.\lambda x.\underline{\text{like'}(y)(x)(w)}$ $\langle e, \langle e, \underline{t}\rangle\rangle$
- $[\![give]\!]^w = \lambda z.\lambda y.\lambda x.\underline{\text{give'}(z)(y)(x)(w)}$ $\langle e, \langle e, \langle e, \underline{t}\rangle\rangle\rangle$

Annotating the trees with types – I

$$\left[\!\!\left[\begin{array}{c} \text{VP} \\ \text{DP} \quad \text{V} \\ \text{Peter}_e \quad \text{sleeps}_{\langle e,t\rangle} \end{array} \right]\!\!\right]^w \overset{2\times(NN)}{=} \left[\!\!\left[\begin{array}{c} \text{VP} \\ \text{Peter}_e \quad \text{sleeps}_{\langle e,t\rangle} \end{array} \right]\!\!\right]^w \overset{(FA)}{=} [\![\text{sleeps}]\!]^w([\![\text{Peter}]\!]^w)$$

$$\left[\!\!\left[\begin{array}{c} \text{VP} \\ \text{V} \quad \text{DP} \\ \text{likes}_{\langle e,\langle e,t\rangle\rangle} \quad \text{Mary}_e \end{array} \right]\!\!\right]^w \overset{2\times(NN)}{=} \left[\!\!\left[\begin{array}{c} \text{VP} \\ \text{likes}_{\langle e,\langle e,t\rangle\rangle} \quad \text{Mary}_e \end{array} \right]\!\!\right]^w \overset{(FA)}{=} [\![\text{likes}]\!]^w([\![\text{Mary}]\!]^w)$$

Formulating (FA) – I

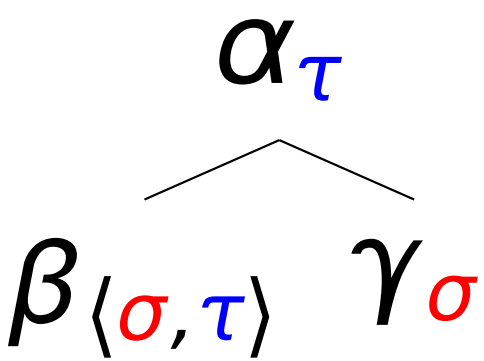
With the help of semantic types, the derivational rule (FA) can be formulated in its full generality.

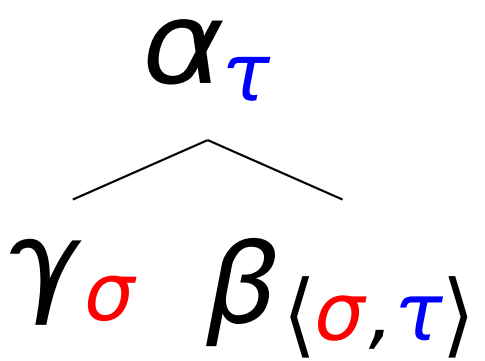
(8) Function application (FA):
For a branching node α with the set of daughters $\{\beta, \gamma\}$, where β is of type $\langle\sigma, \tau\rangle$ and γ is of type σ, then $[\![\alpha]\!]^w = [\![\beta]\!]^w([\![\gamma]\!]^w)$

What is the type of the combination $[\![\beta]\!]^w([\![\gamma]\!]^w)$?

Formulating (FA) – II

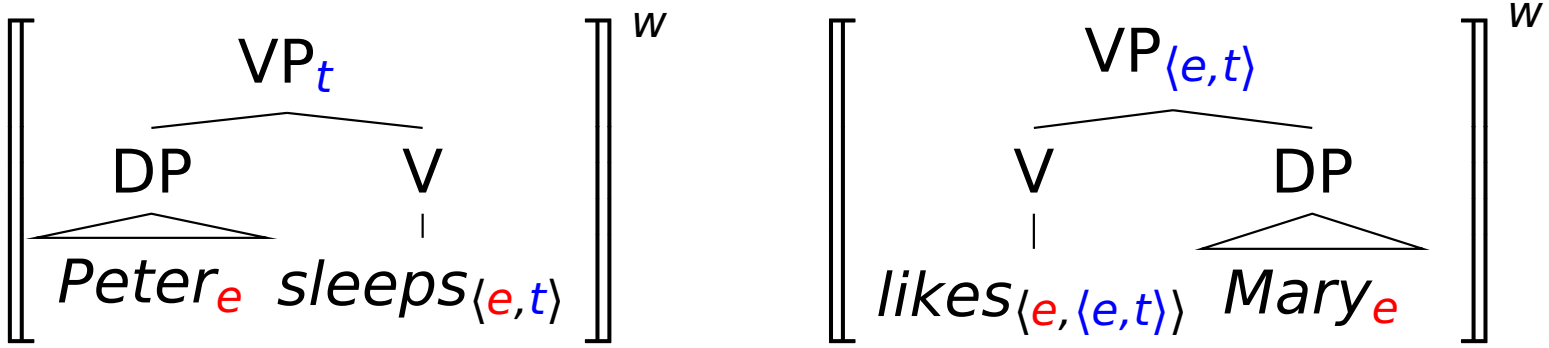
Since the output type of the expression β is τ, the result of $[\![\beta]\!]^w([\![\gamma]\!]^w)$ is of type τ. Given the (FA) rule, the result of $[\![\beta]\!]^w([\![\gamma]\!]^w)$ is $[\![\alpha]\!]^w$ (the extension of the dominating node):





$\Rightarrow$ The logical structure of the output can be determined from the logical structure of the function and the logical structure of the input.

Annotating the trees with types – II



- The type of VP in the left tree is t. This tells us that the composition is finished; all functions have been given all of the arguments they asked for.
- The type of VP in the right tree is $\langle e, t\rangle$. This tells us that composition is not finished; there is still an argument position to be filled.

A different case: transitive verbs – III

What is the result of λ-conversion for the composition of *likes* with *Mary*?

$$\left[\!\left[\begin{array}{c} \mathrm{VP}_{\langle e,t\rangle} \\ \mathrm{V} \quad\quad \mathrm{DP} \\ | \quad\quad\; \diagup\diagdown \\ likes_{\langle e,\langle e,t\rangle\rangle} \quad Mary_e \end{array} \right]\!\right]^w \overset{2\times(NN)}{=} \left[\!\left[\begin{array}{c} \mathrm{VP}_{\langle e,t\rangle} \\ \diagup\diagdown \\ likes_{\langle e,\langle e,t\rangle\rangle} \quad Mary_e \end{array} \right]\!\right]^w$$

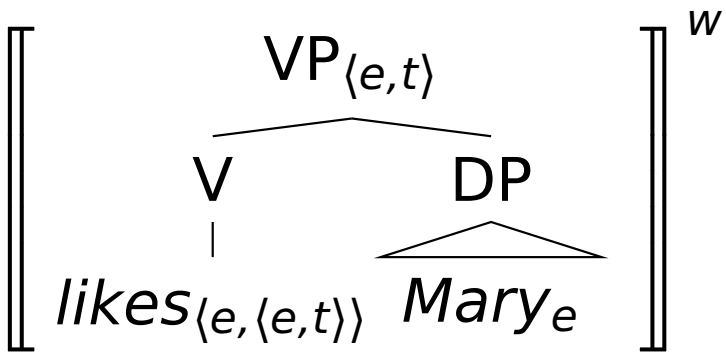
$$\overset{(FA)}{=} [\![likes]\!]^w([\![Mary]\!]^w)$$

$$= [\lambda y.\lambda x.\mathrm{like'}(y)(x)(w)](\mathrm{Mary})$$

$$\overset{\lambda}{=}$$

A different case: transitive verbs – III

What is the result of λ-conversion for the composition of *likes* with *Mary*?



$$\overset{2\times(NN)}{=} \left[\!\!\left[\begin{array}{c} \mathrm{VP}_{\langle e,t\rangle} \\ \diagup\ \diagdown \\ \mathit{likes}_{\langle e,\langle e,t\rangle\rangle}\ \ \mathit{Mary}_e \end{array} \right]\!\!\right]^w$$

$$\overset{(FA)}{=} [\![\mathit{likes}]\!]^w([\![\mathit{Mary}]\!]^w)$$

$$= [\lambda y.\lambda x.\text{like'}(y)(x)(w)](\text{Mary})$$

$$\overset{\lambda}{=} [\lambda x.\text{like'}(\text{Mary})(x)(w)]$$

Summary

- Regarding the realistic view on meaning, sentences have truth values as their extensions. Much more important, however, are the truth conditions of a sentence.
- Since we use language to describe our work on natural language semantics, the distinction between object language and meta language is crucial.
- We formulated the derivational rules (NN) and (FA) which allow us to derive the truth conditions of sentences like '*Peter sleeps*' and '*Peter likes Mary*'.
- To be able to formulate (FA) in its full generality, we introduced semantic types, which encode the logical structure of the extension of an expression.