Introduction to Semantics

Lecture 2: Compositionality and Model Theory

Yasutada Sudo
UCL
y.sudo@ucl.ac.uk

OVA 2026 at Debreceni Egyetem

1 Compositionality

Due to recursion, natural languages contain infinitely many grammatical expressions.

(1) a. It was raining in London.
b. Someone said that it was raining in London.
c. Everyone thought that someone said that it was raining in London.
d. I don't think everyone thought that someone said that it was raining in London.
⋮

(2) a. John never met the student.
b. John never met the student's brother.
c. John never met the student's brother's partner.
d. John never met the student's brother's partner's mother.
e. John never met the student's brother's partner's mother's cousin.
⋮

It is, therefore, impossible to memorize all syntactically grammatical expressions, and we need some generative mechanism that generates all and only grammatical expressions.

On the semantic side, speakers have truth-conditional intuitions about all grammatical sentences, including sentences that they have never encountered before.

Because there are infinitely many grammatical sentences, it must be impossible to make a list of all grammatical sentences and their truth-conditions. Therefore, we need some mechanism that **computes meaning in a systematic and predictable manner** for all grammatical sentences.

- Certainly, the meanings of atomic expressions (= morphemes) are (largely) unpredictable. You need to memorize these.
- But when two words are syntactically combined, the meaning of the resulting expression is (usually) predictable.

(3) a. every
b. a
c. cat
d. dog

(4) a. every cat
b. every dog
c. a cat
d. a dog

Idiomatic expressions are certainly exceptions to the systematicity, but it is undeniable that many syntactically complex expressions have meanings that are predictable from the meanings of the morphemes involved.

The idea that the meaning of a syntactically complex expression can be systematically **computed** from the meanings of its parts is called **compositionality**.

We state this as a principle behind semantic computation (while acknowledging the fact that idioms are exceptions to it).

(5) **Compositionality Principle**: The meaning of a syntactically complex expression is determined by the meanings of its parts and how they are combined (= its syntax).

Exercise: what (non-natural) languages fail to obey this principle?

The relevance of syntax is worth emphasizing. It's easy to see that the syntax matters for the semantics, if you compare sentences like the following.

(6) a. A cat is chasing a dog.
b. A dog is chasing a cat.

(7) a. A cat entered, after a dog entered.
b. A dog entered, after a cat entered.

In other words, the meaning of a syntactically complex expression in natural language is not the result of putting together the meanings of all the morphemes in a blender. Rather, it's necessary to keep track of the syntactic structure of the expression.

Assuming that natural language syntax is always binary-branching, we adopt the following version of the compositionality principle.

(8) **Local Compositionality Principle**: The (truth-conditional) meaning of $\overset{\alpha}{\underset{\beta\ \ \gamma}{\wedge}}$ is determined by the (truth-conditional) meaning of β and the (truth-conditional) meaning of γ and the fact that they are sisters of each other.

In this course, we will only be concerned with truth-conditional meaning.

2 Model-theoretic semantics

The truth-conditional meaning of a sentence is abstract and we cannot really visualize it.

But we can understand it as a *function* that takes a specific scenario and tells you whether

the sentence is true or false.

In the model theoretic approach, we use a mathematical object, called a **model**, instead of an actual scenario.

We will remain implicit about what a model formally is for now, but it is important to remember:

- Each model $\mathcal{M}$ is a mathematical object representing a specific state of affairs.
- Each grammatical expression α is assigned a **model-theoretic object** in $\mathcal{M}$ as its **denotation** with respect to $\mathcal{M}$.
- We refer to the denotation of α with respect to $\mathcal{M}$ by $[\![\alpha]\!]_{\mathcal{M}}$. The subscript $\mathcal{M}$ is often dropped, if it's obvious, but I'll keep it today everywhere just to be explicit.
- The (fragment of the) object language can only talk about model-theoretic objects in $\mathcal{M}$.

2.1 Fragments

What should be in the model? We can talk about all sorts of things in natural language. In fact, it seems that we can somehow talk about most, possibly all, things our brains are capable of thinking about, so much so that it's hard to identify what we can*not* talk about.

But we need to start somewhere, and in order to make the theory manageable, we take the step-by-step approach.

We will start with a small **fragment** of the target language (English), and correspondingly, relatively impoverished models. Specifically, for now, we will only consider declarative sentences that are made up of:

- Singular proper names
- Intransitive and transitive verbs

(9) a. Chris yawned.
b. Katie saw Andrea.

We will ignore the semantic contribution of tense and aspect in this course (although no known language lacks both of these!).

This small fragment will only require a very simple model with:

- **truth-values**
- **individuals**

The model will also contain information about what properties each individual has and what relations the individuals stand in.

In the next lecture we will enrich this fragment with singular count nouns and quantificational determiners, but it turns out that this enrichment will not require a more complex model.

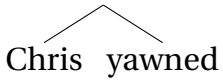
But you might need more things in the model if you want to analyze other expressions:

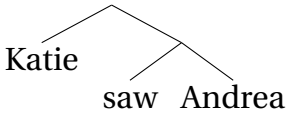
- Non-atomic entities for plural nouns and mass nouns
- Degrees for gradable predicates
- Possible worlds, or situations, for modals
- Time intervals for tense and aspect
- Events for aspect, adverbs

⋮

We will not get to these in this course, but check out the references mentioned in the first handout.

Also, we will assume a very simple syntax like the following.

(10) a. 

b. 

Importantly, it's possible to assume more complex structures for these sentences, although you might have to adjust the details of your semantic analysis accordingly.

We will often omit syntactic labels, since they are, by assumption, void of semantic import. We sometimes add them for expository convenience.

2.2 Declarative sentences denote truth-values

There are two special model-theoretic objects that appear in every model: truth and falsity. They are called **truth-values** and commonly denoted by 1 and 0, respectively (alt.: $\top$, $\bot$).

If sentence ϕ is true in the state of affairs modeled by a model $\mathcal{M}$, we write

$$[\![\phi]\!]_{\mathcal{M}} = 1$$

if it is false in the state of affairs modeled by a model $\mathcal{M}$, we write

$$[\![\phi]\!]_{\mathcal{M}} = 0$$

More concrete examples with hypothetical models $\mathcal{M}_1$ and $\mathcal{M}_2$:

(11) a. $[\![\texttt{Chris likes Alan}]\!]_{\mathcal{M}_1} = 0$
b. $[\![\texttt{Alan likes Chris}]\!]_{\mathcal{M}_1} = 1$
c. $[\![\texttt{Chris likes Alan}]\!]_{\mathcal{M}_2} = 1$
d. $[\![\texttt{Alan likes Chris}]\!]_{\mathcal{M}_2} = 1$
e. $[\![\texttt{David yawned}]\!]_{\mathcal{M}_1} = 0$
f. $[\![\texttt{David yawned}]\!]_{\mathcal{M}_2} = 0$

Some theories have more truth-values, but for now we only assume two truth-values, 1 and 0.

2.3 Proper names denote individuals

Proper names refer to entities. A model $\mathcal{M}$ contains individuals that correspond to them.

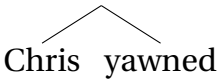
Proper names are analyzed as having such individuals as denotations.

2.4 Intransitive verbs

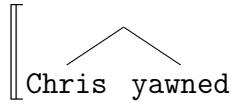
What do intransitive verbs like 'yawn' and 'run' denote? They shouldn't denote truth-values, as they are not true or false on their own. They also don't denote individuals, as they are clearly not referring expressions.

At this point, recall the compositionality principle. Applying it to the denotation of the following example

(12)

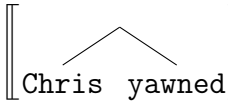


we get (for an arbitrary model $\mathcal{M}$):

(13) $[\![$  $]\!]_{\mathcal{M}}$ is determined by $[\![\texttt{Chris}]\!]_{\mathcal{M}}$ and $[\![\texttt{yawned}]\!]_{\mathcal{M}}$.

We know the left-hand side. It's either the truth-value 1 or the truth-value 0, depending on what $\mathcal{M}$ is like. Similarly, we know $[\![\texttt{Chris}]\!]_{\mathcal{M}}$. It's some individual in $\mathcal{M}$.

Just to be concrete, we can assume a model $\mathcal{M}_5$ where:

(14) a. $[\![$  $]\!]_{\mathcal{M}_5} = 0$ b. $[\![\texttt{Chris}]\!]_{\mathcal{M}_5} = c$

What should $[\![\texttt{yawned}]\!]_{\mathcal{M}_5}$ be?

We can regard it as something that maps c to 0.

Furthermore, we have sentences like (15) for different subject proper names.

(15) a. Anna yawned.
b. Bill yawned.
c. Cathy yawned.
d. David yawned.
e. Eva yawned.
⋮

When a model is given, each of these sentences denotes either 0 or 1. For the individuals that yawned in the model, the corresponding sentences should denote 1 with respect to that model, and for the individuals that did not yawn in the model, the corresponding sentences

should denote 0 with respect to that model.

So taken together, the denotation of 'yawned' in a given model (say, $\mathcal{M}_5$) should combine with an individual and give you 1 if that individual yawned in that model, and 0 if that individual did not yawn in that model.

Because for each individual, it should deterministically say 0 or 1, we can regard it as a function.

(16) $[\![\texttt{yawned}]\!]_{\mathcal{M}_5}$ = the function that takes any individual x in $\mathcal{M}_5$ and returns 1 if x yawned in $\mathcal{M}_5$ and 0 if x did not yawn in $\mathcal{M}_5$.

Keep in mind that this is just a mapping between individuals (in $\mathcal{M}_5$) and truth-values. So another representation of (16) looks like (assuming that a and e yawned in $\mathcal{M}_5$ and c, b and d didn't):

(17) $$[\![\texttt{yawned}]\!]_{\mathcal{M}_5} = \begin{bmatrix} c \mapsto 0 \\ a \mapsto 1 \\ b \mapsto 0 \\ d \mapsto 0 \\ e \mapsto 1 \\ \vdots \end{bmatrix}$$

Either way, it tells you exactly who yawned and who didn't in the given model.

More generally, we can say:

(18) For any model $\mathcal{M}$, $[\![\texttt{yawned}]\!]_{\mathcal{M}}$ = the function that takes any individual x in $\mathcal{M}$ and returns 1 iff x yawned in $\mathcal{M}$.

And we can generalize this to other intransitive verbs:

(19) For any model $\mathcal{M}$,

a. $[\![\texttt{cried}]\!]_{\mathcal{M}}$ = the function that takes any individual x in $\mathcal{M}$ and returns 1 iff x cried in $\mathcal{M}$

b. $[\![\texttt{smokes}]\!]_{\mathcal{M}}$ = the function that takes any individual x in $\mathcal{M}$ and returns 1 iff x smokes in $\mathcal{M}$

c. $[\![\texttt{jumped}]\!]_{\mathcal{M}}$ = the function that takes any individual x in $\mathcal{M}$ and returns 1 iff x jumped in $\mathcal{M}$

This idea enables us to be more concrete about how model-theoretic denotations combine:

(20) For any model $\mathcal{M}$,

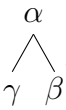
$$\left[\!\!\left[\begin{array}{ccc} & \diagup\diagdown & \\ \texttt{Chris} & & \texttt{yawned} \end{array} \right]\!\!\right]_{\mathcal{M}} = [\![\texttt{yawned}]\!]_{\mathcal{M}}([\![\texttt{Chris}]\!]_{\mathcal{M}})$$

In words, the way the denotation of 'yawned' and that of 'Chris' combine is via **function**

application.

We can assume this as a general **compositional rule**:

(21) **Functional Application**

For any model $\mathcal{M}$, if α is made up of β and γ (that is, either  or  , whichever is grammatical) and $[\![\beta]\!]_{\mathcal{M}}$ is a function such that $[\![\gamma]\!]_{\mathcal{M}} \in \mathrm{dom}\,([\![\beta]\!]_{\mathcal{M}})$, then

$$\left[\!\!\left[\begin{array}{c} \alpha \\ \triangle \end{array} \right]\!\!\right]_{\mathcal{M}} = [\![\beta]\!]_{\mathcal{M}}\,([\![\gamma]\!]_{\mathcal{M}})$$

It should be emphasized that we are computing the denotation/meaning of a syntactically complex expression in terms of the denotations/meanings of its parts (and its syntax). This is more evident in the following example.

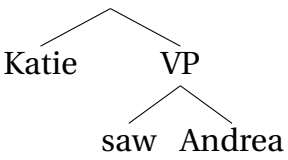
(22)

$$\begin{aligned} \left[\!\!\left[\begin{array}{c} \\ \texttt{Chris}\ \ \texttt{yawned} \end{array} \right]\!\!\right]_{\mathcal{M}_5} &= [\![\texttt{yawned}]\!]_{\mathcal{M}_5}\,([\![\texttt{Chris}]\!]_{\mathcal{M}_5}) \\ &= [\![\texttt{yawned}]\!]_{\mathcal{M}_5}\,(c) \\ &= 0 \end{aligned}$$

2.5 Transitive verbs

Using the same compositional rule, we can figure out the denotations of transitive verbs as well. Let's work with one concrete sentence, (23).

(23)



When a specific model, say $\mathcal{M}_8$, is given, we know the truth-value of this sentence. Let's assume:

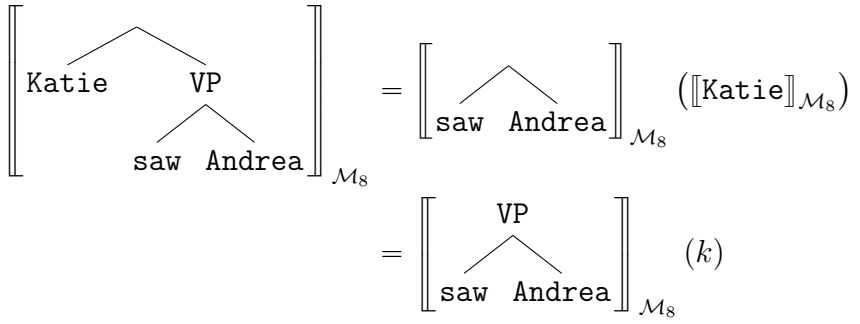
(24) $\left[\!\!\left[\begin{array}{c} \texttt{Katie}\ \ \ \texttt{VP} \\ \texttt{saw}\ \ \texttt{Andrea} \end{array} \right]\!\!\right]_{\mathcal{M}_8} = 1$

The model should also tell you who Katie is. Let's say:

(25) $[\![\texttt{Katie}]\!]_{\mathcal{M}_8} = k$

Assuming that the subject and the VP combine via the compositional rule of Functional Application, we have:

(26)



By the same reasoning as in the case of intransitive verbs, we want the VP denotation to be something that maps k to 1. And because the subject can be any proper name, we generally want the VP denotation to map any individual in the model to a truth-value depending on whether or not they saw the individual that the proper name 'Andrea' refers to.

(27) For any model $\mathcal{M}$,
$\left[\!\left[\text{VP} \atop \text{saw Andrea} \right]\!\right]_{\mathcal{M}}$ = the function that takes any individual x in $\mathcal{M}$ and returns 1 iff x saw $[\![\text{Andrea}]\!]_{\mathcal{M}}$ in $\mathcal{M}$.

Applying the same reasoning again, we can deduce the denotation of 'saw' alone. That is, abstracting over the object position, we want for any proper name α,

(28) For any model $\mathcal{M}$, $\left[\!\left[\text{VP} \atop \text{saw } \alpha \right]\!\right]_{\mathcal{M}}$ = the function that takes any individual x in $\mathcal{M}$ and returns 1 iff x saw $[\![\alpha]\!]_{\mathcal{M}}$ in $\mathcal{M}$.

Now, assuming that 'saw' and α combine via Functional Application, we can conclude:

(29) For any model $\mathcal{M}$, $[\![\text{saw}]\!]_{\mathcal{M}}$ is the function that takes any individual y in $\mathcal{M}$ and returns the function that takes an individual x in $\mathcal{M}$ and returns 1 iff x saw y in $\mathcal{M}$.

Assuming, for example, $[\![\text{Andrea}]\!]_{\mathcal{M}_8} = a$,

(30) $\left[\!\left[\text{VP} \atop \text{saw Andrea} \right]\!\right]_{\mathcal{M}_8} = [\![\text{saw}]\!] \left([\![\text{Andrea}]\!]_{\mathcal{M}_8}\right)$
$= [\![\text{saw}]\!]_{\mathcal{M}_8}(a)$
$=$ the function that takes any individual x in $\mathcal{M}_8$ and returns 1 iff x saw a in $\mathcal{M}_8$

(31) $\left[\!\left[\begin{array}{c} \texttt{Katie} \quad \texttt{VP} \\ \qquad\quad \texttt{saw} \quad \texttt{Andrea} \end{array}\right]\!\right]_{\mathcal{M}_8} = \left[\!\left[\begin{array}{c} \texttt{VP} \\ \texttt{saw} \quad \texttt{Andrea} \end{array}\right]\!\right]_{\mathcal{M}_8} \left(\llbracket \texttt{Katie} \rrbracket_{\mathcal{M}_8}\right)$

$= \left[\!\left[\begin{array}{c} \texttt{VP} \\ \texttt{saw} \quad \texttt{Andrea} \end{array}\right]\!\right]_{\mathcal{M}_8} (k)$

$=$ [the function that takes any individual x in $\mathcal{M}_8$ and returns 1 iff x saw a in $\mathcal{M}_8$](k)

$= 0$

We can analyze other transitive verbs in the same way:

(32) For any model $\mathcal{M}$,

a. $\llbracket \texttt{likes} \rrbracket_{\mathcal{M}}$ is the function that takes any individual y in $\mathcal{M}$ and returns the function that takes any individual x in $\mathcal{M}$ and returns 1 iff x likes y in $\mathcal{M}$

b. $\llbracket \texttt{invited} \rrbracket_{\mathcal{M}}$ is the function that takes any individual y in $\mathcal{M}$ and returns the function that takes any individual x in $\mathcal{M}$ and returns 1 iff x called y in $\mathcal{M}$

3 λ-notation for functions

In formal semantics, it is customary to use **lambda-notation** for functions.

In the standard lambda-notation used in formal semantics,[1] a function generally looks like:

$$\lambda v\colon\ \phi.\ \alpha$$

- The symbol λ (the Greek letter 'lambda') has no meaning on its own. It's there to indicate that this whole formal expression names a function.
- v is a variable, representing the input of the function. It's a variable because its value varies depending on what the input is. You are allowed to pick any variable name, but we will use x, y, f, P, etc. in this couse.
- ϕ is a statement describing what kind of input the function admits (typically in terms of v). This means that ϕ defines the domain of the function. Keep in mind that ϕ needs to be a statement that can be true or false. The idea is that if ϕ is true, the input is an appropriate kind of object for the function to operate on, and if ϕ is false, it is not and the function will not output anything.
- α describes the output, often in terms of v.

It's easier to understand this by looking at concrete examples. Here is one:

(33) $\lambda x\colon\ x$ is a natural number. $x + 5$

This is a function that takes a natural number and adds 5 to it. You are probably more familiar with the notation $f(x) = x + 5$. (33) is exactly the same function.

[1] Linguists borrowed this notation from lambda calculus, a formal system for functions developed by Alonzo Church.

You might wonder why we need another way of writing the same thing. Lambda-notation becomes particularly convenient for us, because it makes it clear that you are referring to the function itself, rather than the output value of f applied to x (which we will consistently denote by '$f(x)$'). Also, in lambda-notation the domain of the function can be explicitly stated (e.g, the part that says 'x is a natural number' in the above example).

You can also write the same function more compactly as (34). $\mathbb{N}$ is the symbol that is commonly used to denote the set of natural numbers.

(34) $\lambda x\colon\ x \in \mathbb{N}.\ x + 5$

Functions like (34) whose domain is expressed as the variable x being a member of some set S are often abbreviated to '$\lambda x \in S.\ \alpha$':

(35) $\lambda x \in \mathbb{N}.\ x + 5$

In this representation, you no longer see ':', but remember that there is an implicit ':'.

The choice of the variable x above is not important, because it is a variable. We could have chosen, say, y instead, as in (36), and meant the same thing.

(36) $\lambda y \in \mathbb{N}.\ y + 5$

Usually, we use x, y, z as variable names, especially when they stand for arbitrary individuals. In principle they can be any expression (as long as you declare that it is a variable), but you should stick to standard ones for readability's sake.

Functions need not be about mathematical operations. For instance, (37) is a legitimate function (assuming that each dog has a unique name).

(37) $\lambda y\colon\ y$ is a dog in London. y's name

This is a function that takes any dog in London and returns its name.

Using lambda-notation, we can re-write the denotations of the intransitive verbs:

(38) For any model $\mathcal{M}$,
   a. $[\![\texttt{yawned}]\!]_{\mathcal{M}} = \lambda x\colon\ x$ is an individual in $\mathcal{M}$. 1 iff x yawned in $\mathcal{M}$
   b. $[\![\texttt{cried}]\!]_{\mathcal{M}} = \lambda x\colon\ x$ is an indvidual in $\mathcal{M}$. 1 iff x cried in $\mathcal{M}$

The set of all entities in a given model (called the domain of the model) is often written as D (with the model name being implicit; if you want to be explicit, you can write $D_{\mathcal{M}}$ for model $\mathcal{M}$). So, these functions are more compactly written as:

(39) For any model $\mathcal{M}$,
   a. $[\![\texttt{yawned}]\!]_{\mathcal{M}} = \lambda x \in D.\ 1$ iff x yawned in $\mathcal{M}$
   b. $[\![\texttt{cried}]\!]_{\mathcal{M}} = \lambda x \in D.\ 1$ iff x cried in $\mathcal{M}$

From now on, we will use these representations.

(If you read published papers in formal semantics, '1 iff' is often also omitted, but we will keep it here, because this convention can be confusing at the beginning.)

3.1 Functions that return functions

Things get a bit more complicated, when you consider functions that return functions as values, but the basic principles are the same.

E.g., we can define a function A that takes a natural number n, and returns a function, which in turn takes a natural number m and returns the value $n + m$. As you can see, A represents addition, but it takes one number at a time. In particular, when you feed A with one natural number, you will get another function. For example, $A(3)$ and $A(5)$ will be both functions, but crucially, $A(3) \neq A(5)$, because $A(3)$ is the function that takes a natural number and adds 3 to it, while $A(5)$ is the function that takes a natural number and adds 5 to it. In λ-notation:

(40) a. $A(3) = \lambda m \in \mathbb{N}.\ m + 3$
b. $A(5) = \lambda m \in \mathbb{N}.\ m + 5$

λ-notation allows you to represent A itself as well. It will look like (41). We will use '[' and ']' to indicate where each function starts and ends.

(41) $A = [\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n + m]]$

This representation of A looks as if A has two λ's, but that's not the right way of understanding it. Recall that in λ-notation, each function has three parts (apart from λ). The first two parts are collapsed into '$\lambda n \in \mathbb{N}$' here, which says that it takes as input any member n of the set $\mathbb{N}$, which is to say that it takes any natural number n. Then the output description specifies what function it turns, namely $[\lambda m \in \mathbb{N}.\ n + m]$. So, if $n = 3$, it returns $[\lambda m \in \mathbb{N}.\ 3 + m]$, if $n = 5$, it returns $[\lambda m \in \mathbb{N}.\ 5 + m]$, and so on. These resulting functions in turn have three parts. They take any member m of $\mathbb{N}$, i.e. any natural number, and returns $3 + m$, $5 + m$, etc.

Using this, we can re-write the denotations of the transitive verbs.

(42) For any model $\mathcal{M}$,
a. $[\![\texttt{saw}]\!]_{\mathcal{M}} = [\lambda x \in D.\ [\lambda y \in D.\ 1 \text{ iff } y \text{ saw } x \text{ in } \mathcal{M}]]$
b. $[\![\texttt{likes}]\!]_{\mathcal{M}} = [\lambda x \in D.\ [\lambda y \in D.\ 1 \text{ iff } y \text{ likes } x \text{ in } \mathcal{M}]]$
c. $[\![\texttt{invited}]\!]_{\mathcal{M}} = [\lambda x \in D.\ [\lambda y \in D.\ 1 \text{ iff } y \text{ likes } x \text{ in } \mathcal{M}]]$

3.2 λ-conversion

When a function applies to an appropriate argument, you can simplify the representation. This process is often called 'λ-conversion' (or β-reduction). Here is an example of

$[\![\texttt{smokes}]\!]_{\mathcal{M}}$ applied to c.

$$\begin{aligned}[\![\texttt{smokes}]\!]_{\mathcal{M}}(c) &= [\lambda x \in D.\ 1 \text{ iff } x \text{ smokes in } \mathcal{M}](c)\\ &= 1 \text{ iff } c \text{ smokes in } \mathcal{M}\end{aligned}$$

Recall that we are following the standard convention that the function comes to the left of the argument and the argument is indicated by '(' and ')'. Generally, for any function f and for any argument a for it, we write '$f(a)$' to denote the result of applying f to a. So things like '$(c)[\lambda x \in D.\ 1 \text{ iff } x \text{ smokes in } M]$' and '$(\lambda x \in D.\ 1 \text{ iff } x \text{ smokes in } M)[c]$' don't make sense. Such expressions are ungrammatical in our metalanguage, and should never be used.

Here are some more examples of λ-conversion.

(43) $[\lambda x \in \mathbb{N}.\ x + 5](12) = 12 + 5 = 17$

(44) $[\lambda x \in \mathbb{N}.\ x + 5]([\lambda y \in \mathbb{N}.\ y^2](3)) = [\lambda x \in \mathbb{N}.\ x + 5](3^2) = [\lambda x \in \mathbb{N}.\ x + 5](9) = 9 + 5 = 14$

Recall that a function can return a function. Consider the following function S. This function performs subtraction.

(45) $S = [\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n - m]]$

When one argument, say 3, is given, it returns the following function.

$$S(3) = [\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n - m]](3) = [\lambda m \in \mathbb{N}.\ 3 - m]$$

And here is a particularly important convention. When two arguments follow a function, the function is applied to the left one (= the one closer to the function) first.

$$S(3)(2) = [\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n - m]](3)(2) = [\lambda m \in \mathbb{N}.\ 3 - m](2) = 3 - 2 = 1$$

Notice that $S(2)(3) = -1$, so $S(3)(2) \neq S(2)(3)$, meaning that it matters (in this case) which argument comes first.

Also remember that square brackets are used to delimit functions. Notice the following inequality:

$$[\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n - m]](3)(2) \neq [\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n - m](3)](2)$$

The left-hand side is 1, as calculated above. The right-hand side is -1, because:

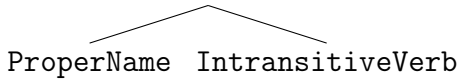
$$[\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n - m](3)](2) = [\lambda n \in \mathbb{N}.\ n - 3](2) = 2 - 3 = -1$$

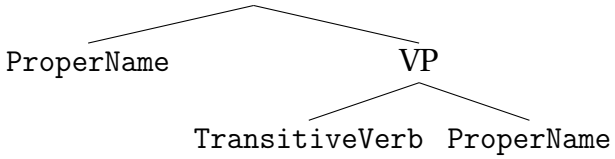
Here, 3 is the argument of the inner function, and 2 is the argument of the outer function that contains the inner function. Here we applied the inner function to 3 first, but generally, the order of application does not change the final output (a property called the Church-Rosser property).

$$[\lambda n \in \mathbb{N}.\ [\lambda m \in \mathbb{N}.\ n - m](3)](2) = [\lambda m \in \mathbb{N}.\ 2 - m](3) = 2 - 3 = -1$$

4 The fragment so far

Our fragment so far is very small, only containing declarative sentences of the form:

(46) a. 

b. 

We assume that proper names always denote individuals (if they have referents in the given model).

Intransitive and transitive verbs denote functions.

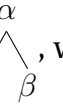
(47) For any model $\mathcal{M}$,
- a. $[\![\texttt{yawned}]\!]_{\mathcal{M}} = [\lambda x \in D.\ 1 \text{ iff } x \text{ yawned in } \mathcal{M}]$
- b. $[\![\texttt{cried}]\!]_{\mathcal{M}} = [\lambda x \in D.\ 1 \text{ iff } x \text{ cried in } \mathcal{M}]$
- c. $[\![\texttt{smokes}]\!]_{\mathcal{M}} = [\lambda x \in D.\ 1 \text{ iff } x \text{ smokes in } \mathcal{M}]$
- d. $[\![\texttt{jumped}]\!]_{\mathcal{M}} = [\lambda x \in D.\ 1 \text{ iff } x \text{ jumped in } \mathcal{M}]$

(48) For any model $\mathcal{M}$,
- a. $[\![\texttt{saw}]\!]_{\mathcal{M}} = [\lambda x \in D.\ [\lambda y \in D.\ 1 \text{ iff } y \text{ saw } x \text{ in } \mathcal{M}]]$
- b. $[\![\texttt{likes}]\!]_{\mathcal{M}} = [\lambda x \in D.\ [\lambda y \in D.\ 1 \text{ iff } y \text{ likes } x \text{ in } \mathcal{M}]]$
- c. $[\![\texttt{invited}]\!]_{\mathcal{M}} = [\lambda x \in D.\ [\lambda y \in D.\ 1 \text{ iff } y \text{ likes } x \text{ in } \mathcal{M}]]$

We can regard these as part of the lexical knowledge about these words.

These semantic ingredients combine via the compositional rule of Functional Application.

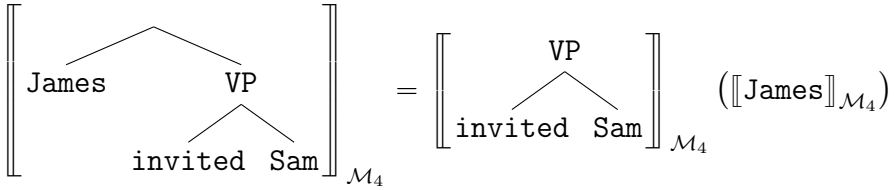
(49) **Functional Application**

For any model $\mathcal{M}$, if α is made up of β and γ (that is, either  or  , whichever is grammatical) and $[\![\beta]\!]_{\mathcal{M}}$ is a function such that $[\![\gamma]\!]_{\mathcal{M}} \in \text{dom}\,([\![\beta]\!]_{\mathcal{M}})$, then

$$\left[\!\!\left[\begin{array}{c} \alpha \\ \triangle \end{array} \right]\!\!\right]_{\mathcal{M}} = [\![\beta]\!]_{\mathcal{M}}\,([\![\gamma]\!]_{\mathcal{M}})$$

Although we do not have infinitely many sentences in this fragment (due to the lack of recursion), our semantic theory obeys the compositionality principle.

Example semantic computation: Let's pick a specific model, $\mathcal{M}_4$.

(50) 

$= [\![\texttt{invited}]\!]_{\mathcal{M}_4} \left([\![\texttt{Sam}]\!]_{\mathcal{M}_4}\right) \left([\![\texttt{James}]\!]_{\mathcal{M}_4}\right)$
$= [\lambda x \in D.\, [\lambda y \in D.\, 1 \text{ iff } y \text{ invited } x \text{ in } \mathcal{M}_4]] \left([\![\texttt{Sam}]\!]_{\mathcal{M}_4}\right) \left([\![\texttt{James}]\!]_{\mathcal{M}_4}\right)$
$= [\lambda y \in D.\, 1 \text{ iff } y \text{ invited } [\![\texttt{Sam}]\!]_{\mathcal{M}_4} \text{ in } \mathcal{M}_4] \left([\![\texttt{James}]\!]_{\mathcal{M}_4}\right)$
$= 1 \text{ iff } [\![\texttt{James}]\!]_{\mathcal{M}_4} \text{ invited } [\![\texttt{Sam}]\!]_{\mathcal{M}_4} \text{ in } \mathcal{M}_4$

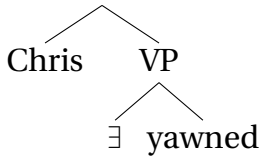
The exact truth-value depends on what $\mathcal{M}_4$ is like. Note that it's not that we wanted to figure out the truth-value by the above computation; we have the truth-conditional intuition anyway. Rather, we wanted to demonstrate that the truth-value of the sentence is computed from the denotations of its parts, and we have achieved that.

5 A quick note about events

You might think that verbs should be about events (or 'eventualities', more generally), rather than about individuals. This idea is actually not incompatible with the toy theory we have developed. In order to have events in our theory, we will have to complicate the syntax a bit more.

One possible way of doing so would involve a phonologically null existential quantifier for eventualities, call it ∃, in the following manner:

(51)

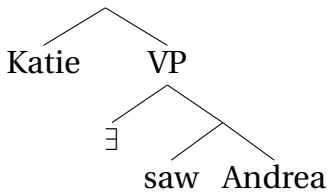


Our analysis of 'yawned' as a function from individuals in the model to truth-values can actually maintained as is with this new syntax, except that it is now the analysis of the VP node, which is syntactically complex. Because it is syntactically complex, you can analyze it further, using a compositional rule.

I believe Berit will discuss Event Semantics in Week 2, so I will not delve into this any further here.

In the case of a transitive verb, the new analysis will not involve a node that denotes the same function from individuals to functions from individuals to truth-values.

(52)



However, the VP node will still denote the same function from individuals to truth-values as before. And more importantly, the way you develop a semantic analysis will stay the same.

You could similarly postulate further nodes in the syntactic structure nodes that have to do with tense and aspect, and adjust your semantic analysis accordingly. That's one way of extending your fragment to more linguistic phenomena.