Intro to formal semantics. Part 2. Ekaterina Vostrikova
EGG 2024 in Brașov: July 22-August 2

https://www.vostrikova.info/teaching

Lecture 1-2

1. Rules of interpretation

1.1 Recap from Part 1 of this course and some slight changes in our notation

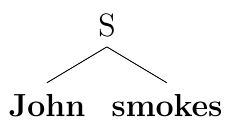
1.1.1 Intransitive verbs

(1)

Compositional Determination of the Extension of Subject-Predications
If S is a sentence with a predicate P and a proper name NN as its subject, the for all $s \in LS$ the following holds:
$[\![S]\!]^s = [\![P]\!]^s([\![NN]\!]^s)$

This rule is designed for sentences of this shape:

(2)



(3) $[\![\textbf{smokes}]\!]^s = \lambda y.\ \vdash y \text{ smokes in } s \dashv$

(4) $[\![\textbf{John}]\!]^s = \text{John}$

(5) $[\![\textbf{John smokes}]\!]^s = \vdash \text{John smokes in } s \dashv$

(6)

D2.2 If φ is a statement, then $\vdash\varphi\dashv$ is the truth value of φ; i.e.:

- $\vdash\varphi\dashv = 1$ if φ is the case; and
- $\vdash\varphi\dashv = 0$ otherwise.

I am going to use a slightly different notation to represent <u>the same thing</u>.

(7) $[\![\textbf{smokes}]\!]^s = \lambda y.$ y smokes in s

(8) $[\![\textbf{John smokes}]\!]^s$ = T iff John smokes in s

or

(9) $[\![\textbf{John smokes}]\!]^s$ = 1 iff John smokes in s

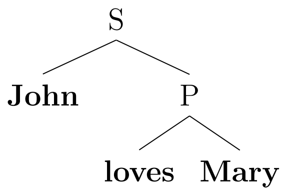
1.1.2 Transitive verbs

(10)

Compositional Determination of the Extensions of Direct-Object-Predications
If P is a predicate consisting of a transitive verb V and a proper name NN as its direct object, then for all $s \in LS$ the following holds:
$[\![P]\!]^s = [\![V]\!]^s([\![NN]\!]^s)$.

This rule is designed for sentences of this shape:

(11)



(12) $[\![\textbf{loves}]\!]^s = [\lambda x.[\lambda y.$ y loves x in s$]]$

(13) $[\![\textbf{loves Mary}]\!]^s =$
$[\![\textbf{loves}]\!]^s\ ([\![\textbf{Mary}]\!]^s) =$
$[\lambda x.[\lambda y.$ y loves x in s$]]$ (Mary) $=$
$[\lambda y.$ y loves Mary in s$]$

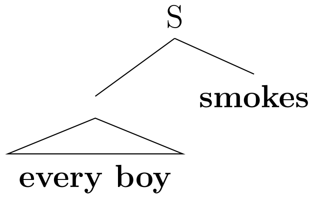
1.1.3 Quantifiers

(14)

Compositional Determination of the Extension of Subject-Quantifications
If S is a sentence with a predicate P and a quantifying noun phrase QN as its subject, then the following holds for all $s \in LS$:
$[\![S]\!]^s = [\![QN]\!]^s([\![P]\!]^s)$.

This rule is designed for sentences of this shape:

(15)



There is a missed generalization here!

Every time when one expression was a function, and another one could potentially be its argument, we interpreted the structure by applying the result of interpretation of the expression that denotes a function to an expression that denotes its potential the argument.

1.2. The Rule of Functional Application

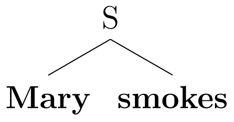
What we now need is a *rule that would allow us to compose the denotation of the two daughter nodes* in order to get the denotation of the whole sentence together with its truth conditions.

This is the first rule of semantic composition that we will introduce. It is called the Rule of Functional Application (FA) [Heim and Kratzer 1998: 44]:

(16) **Functional Application**: If α is a branching node that has two daughters — β and γ — and if $\llbracket\beta\rrbracket$ is a function whose domain is $\llbracket\gamma\rrbracket$, then $\llbracket\alpha\rrbracket = \llbracket\beta\rrbracket(\llbracket\gamma\rrbracket)$.

Let's look at a couple of toy examples:

(17)



- the function, which is the denotation of **smokes**, applies to the individual Mary, which is the denotation of **Mary**, and outputs 1 iff Mary smokes.

(18) $\llbracket\textbf{S}\rrbracket^{s}$ = by FA

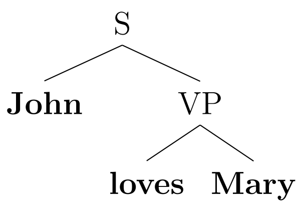
$\llbracket\textbf{smokes}\rrbracket^{s}(\llbracket\textbf{Mary}\rrbracket^{s})$ = by lexicon

λy. y smokes in s (Mary) = by lambda conversion and by our convention

T iff Mary smokes

Notice that FA does not care about the relative order of the function and the argument in the tree[1].

(19)



(20) ⟦VP⟧^{s} = by FA
⟦**loves**⟧^{s} (⟦**Mary**⟧^{s}) = by lexicon
[λx.[λy. y loves x in s]] (Mary) = by lambda conversion
[λy. y loves Mary in s]

A side note on lambda conversion:

Step 1: find the closest closing square bracket:
[λx.[λy. y loves x in s]] (Mary)

Step 2: delete:
the lambda term
the variable following it
the outer layer of the square brackets
put Mary exactly at the place where that same variable was before:

[λy. y loves Mary in s]

From now on I will ignore s subscript on the interpretation function.
So instead of writing ⟦loves⟧^{s}, I am just going to write ⟦loves⟧.
Accordingly, I will also not write [λy. y loves Mary in s], I will just write [λy. y loves Mary] until we will actually need intensions.

1.3. Semantic types and type driven interpretation

We saw that:

- Proper names denote individual objects also known as *entities*.
- Declarative sentences denote truth values.

[1] I will use other labels for certain nodes than the ones Ede was using. The labels I will use are the ones familiar from a syntax class. You can use any labels you prefer; semantics is blind to this.

Then, we also saw that:

- Intransitive verbs denote functions from entities to truth values.
- Transitive verbs denote functions from entities to functions from entities to truth values

It is convenient at this point to systematize and label these types of denotations.

Semantic types:
e is the type of individuals
t is the type of truth-values.

In addition to these basic types there derived types for functions.

These are labeled by ordered pairs $<\sigma,\tau>$, where the first element stands for the type of the argument of a function, the second the type of values of this function have.

In general, $D\tau$ is the set of possible denotations of type τ

The list of possible denotation types we have so far:

(21)
a. The domain of entities D_e
$D_e = \{x: x \text{ is an entity}\}$

b. The domain of truth values D_t
$D_t = \{0,1\}$

c. The domain of functions from entities to truth values $D_{<e,t>}$
$D_{<e,t>} = \{f: f \text{ is from } D_e \text{ to } D_t\}$

d. The domain of functions from entities to function from entities to truth values
$D_{<e,<e,t>>}$
$D_{<e,<e,t>>} = \{f: f \text{ is from } D_e \text{ to } D_{<e,t>}\}$

In a corresponding manner, we will speak about semantic types of expressions

(22)

a. Expressions like **Mary**, which denote entities, are expressions of *semantic type* e.

b. Expressions like **Mary smokes**, which denote truth values, are expressions of *semantic type* t.

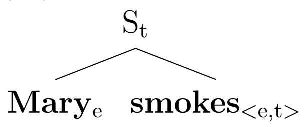
c. Expressions like **smokes** are of type <e,t> (which means they denote functions whose arguments are of type e, and whose values are of type t)

d. Expressions like **loves are** of type <e,<e,t>> (which means they denote functions whose arguments are of type e, and whose values are functions of type <e,t>)

Semantic types of expressions thus reflect the type of the denotations of those expressions.

We can label the expressions in our toy tree with subscripts representing their semantic types:

(23)



The labels now reflect which expression denotes the function, which expression denotes its argument, and which expression denotes the value.

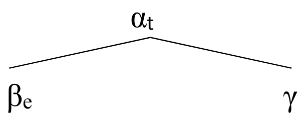
The denotation of the mother node is thus entirely determined by the denotation of its daughters (β and γ) and their mode of composition.

This is the principle of type-driven interpretation and it applies to all other configurations that we'll look at.

This principle also helps us find the semantic type of a daughter if we know the semantic types of its sister and mother nodes.

In the tree below, we see that under the type-driven interpretation approach, the only type that expression γ can be is <e,t>.

(24)

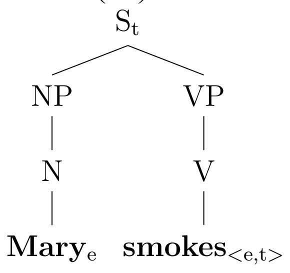


1.4. Rules for Terminal and Non-branching nodes

The toy tree we looked at before is not something that we normally work with.

A more realistic syntactic tree for **Mary smokes** looks as:

(25)



This is an alternative representation of this tree that you will see used a lot in the literature.

(26) $[_S\ [_{NP}\ [_N \text{ Mary}]]\ [_{VP}\ [_V \text{ smokes}]]]$

Here, again our goal is to see how the interpretation of the whole sentences is derived compositionally from the interpretation of its constituent parts.

We thus need to be able to assign a denotation to each node in the tree and make sure that the denotation of each node is derived compositionally.

We begin with the terminal nodes (the leaves of the syntactic tree).

Above (when we looked at our toy tree) we simply assumed that the interpretation of the terminal nodes is the interpretation of the word.

Now, we state it formally as a rule:

(27) **The <u>Terminal Nodes Rule</u> (TN):**
If α is a terminal node, $[\![\alpha]\!]$ is specified in the lexicon.

What about the non-branching nodes N and V and, consequently, NP and VP?

They are not part of the lexicon. To get the interpretation of these nodes, we employ the Rule of non-branching nodes:

(28) **The <u>Non-Branching Nodes Rule</u> (NB):**
If α is a non-branching node and β is α's daughter, then $[\![\alpha]\!] = [\![\beta]\!]$.

The mother and the daughter in a non-branching node are of the same semantic type.

2. Quantifier: their meaning and semantic type

2.1 Introduction

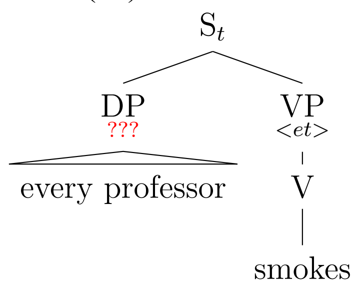
Last time, Ede talked about **quantifiers**.

Here are some examples:

(29)	A/some professor	**A / some professor** smokes.
(30)	No professor	**No professor** smokes.
(31)	Every professor	**Every professor** smokes.
(32)	Three professors	**Three professors** smoke.
(33)	Many professors	**Many professors** smoke.
(34)	Few professors	**Few professors** smoke.
(35)	Most professors	**Most professors** smoke.

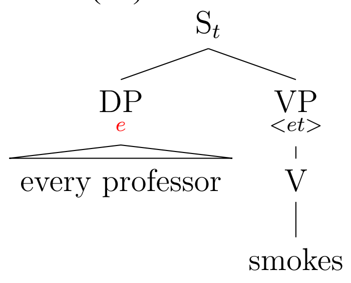
What is the semantic type of these expressions?

(36)



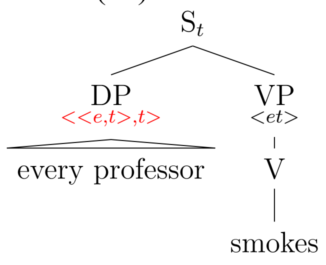
Option 1

(37)



Option 2

(38)



We are going to see some arguments in favour of Option 2 and against Option 1.

2.2 Option 2 is the way to go!

2.2.1 Argument 1

- Suppose that we have two VPs – VP_1 and VP_2 – such that for all x, if ⟦$\mathbf{VP_1}$⟧(x) = T then ⟦$\mathbf{VP_2}$⟧(x) = T.
- For example for all x, if ⟦**smokes Marlboros**⟧(x) = T, then ⟦**smokes**⟧(x) = T
- Let's consider a DP of type e, say 'Chomsky'
- It follows that if ⟦ **DP** $\mathbf{VP_1}$⟧ = T, then ⟦ **DP** $\mathbf{VP_2}$ ⟧ = T.
- ⟦Chomsky smokes Marlboros⟧ = T then ⟦ Chomsky smokes⟧ = T.

The following, however, does not hold:

- if ⟦**no professor** smokes Marlboros⟧ = T, then ⟦**no professor** smokes⟧ = T

The DP *no professor* is **not of type e!**

2.2.2 Argument 2

- Suppose that we have a VP_1 and VP_2 – such that VP_2 is formed by adding 'does not' to VP_1
- Then, for any x, ⟦$\mathbf{VP_1}$⟧(x) = T iff ⟦$\mathbf{VP_2}$⟧(x) = F
- For example, for any x, ⟦**smokes**⟧(x) = T iff ⟦**does not smoke**⟧(x) = F
- Let's consider a DP of type e, say 'Chomsky'
- ⟦**Chomsky smokes**⟧ = T *iff* ⟦**Chomsky doesn't smoke**⟧ = F

The following DOESN'T hold:

⟦**a/some professor** smokes⟧ = T *iff* ⟦**a/some professor doesn't smoke**⟧ = F

The following can be the case:

⟦**a/some professor** smokes⟧ = T *and* ⟦**a/some professor doesn't smoke**⟧ = T

The DP *a / some professor* is **not of *type e.***

2.2.3 Argument 3

- Let's again consider two VPs: VP_1 and VP_2 – such that VP_2 is formed by adding 'does not' to VP_1
- Then, for any DP of type e ⟦DP $\mathbf{VP_1}$⟧ or ⟦DP $\mathbf{VP_2}$⟧ is necessarily true.
- For example 'Chomsky smokes or Chomsky does not smoke' is necessarily true.
- We cannot find a scenario/construct a situation when this could be false.
- Now let's look at 'every professor'

- 'Every professor smokes or every professor does not smoke' is not necessarily true
- We can find a scenario/ construct a situation when this is false.
    - For example, imagine we have 3 professors: A, B and C.
    - Imagine A smokes, B and C do not smoke.
    - Then it is false that every professor smokes.
    - And it is false that every professor does not smoke.
    - Therefore, it is false that every professor smokes or every professor does not smoke
- The DP *every professor* is **not of *type e*.**

2.2.4 Argument 4

All sentences below mean the same thing:

(39) John saw Mary.
(40) John is such that he saw Mary.
(41) Mary is such that John saw her.

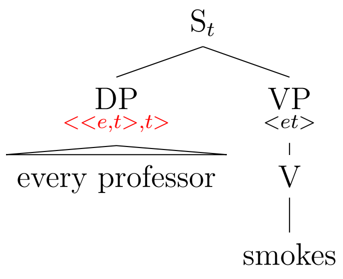
This does not hold for the following:

(42) Some student saw every professor.

(43) Some student is such that he saw every professor.
This requires that one student, say, John saw every professor.

(44) Every professor is such that some student saw her.
This could be true if for every professor there is a different student who saw her.

- DPs like 'every professors', 'some professor', 'no professor' are not of *type e*:
- Therefore, they must be of type <<e,t>,t>
- We call DPs denoting such functions quantificational DPs
- We also use the term 'Generalized quantifiers'



2.3 The Semantics of Quantificational DPs

- Now let's see what kind of function a quantificational DP denote.

- As an <<e,t>,t> function, the extension of a quantificational DP takes an <e,t> as its argument and returns a truth value.

- These DPs denote predicates of predicates (or 'second order predicates/properties)

- They 'say things about' their <e,t> arguments.

    - "No professor" says that its VP argument is true of no professor

(45) ⟦**No professor**⟧(⟦**VP**⟧) = T *iff* there is no professor x such that ⟦**VP**⟧(x) = T
(46) ⟦**No professor**⟧(⟦**smokes**⟧) = T *iff* there is no professor x s.t. ⟦**smokes**⟧(x) = T

- "A / Some professor" says that its VP argument is true of some professor

(47) ⟦**A/Some professor**⟧(⟦VP⟧) = T *iff* there is some professor x such that ⟦VP⟧(x) = T
(48) ⟦**A professor**⟧(⟦**smokes**⟧) = T *iff* there is some professor x such that ⟦**smokes**⟧(x) = T

- "Every professor" says that its VP argument is true of every professor

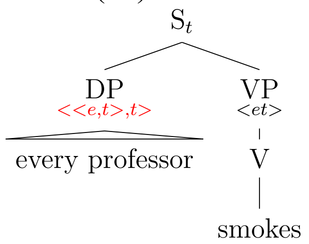
(49) ⟦**Every professor** ⟧(⟦**VP**⟧) = T *iff* for all x, if x is a professor, then ⟦VP⟧(x) = T
(50) ⟦**Every professor**⟧(⟦**smokes**⟧) = T *iff* for all x, if x is a professor, then ⟦smokes⟧(x) = T

(51) ⟦**no professor**⟧ = [$\lambda f_{<e,t>}$. there is no professor x such that f(x) = T]
(52) ⟦**a/some professor**⟧ = [$\lambda f_{<e,t>}$ .there is some professor x such that f(x) = T]
(53) ⟦**every professor**⟧ = [$\lambda f_{<e,t>}$. for all x, if x is a professor, then f(x) = T]

Another notation that is often used:

(54) ⟦**no professor**⟧ = [$\lambda f_{<e,t>}$. $\neg\exists$x[x is a professor & f(x) = T]
(55) ⟦**a/some professor**⟧ = [$\lambda f_{<e,t>}$.$\exists$x[x is a professor & f(x) = T]
(56) ⟦**every professor**⟧ = [$\lambda f_{<e,t>}$.$\forall$x[x is a professor$\rightarrow$ f(x) = T]

,

(57)


The derivation:

(58)

⟦**S**⟧ = by FA

⟦**DP**⟧ (⟦**VP**⟧)= by 3 applications of NN

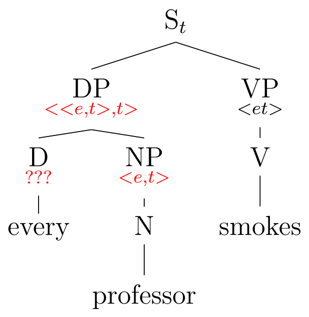
⟦**every professor**⟧ (⟦**smokes**⟧) = by TN and the lexicon

[$\lambda f_{<et>}$. for all x, if x is a professor, then f(x) = T] (λy_e.y smokes) =

T iff for all x, if x is a professor, then [λy_e.y smokes](x) =T =

T iff for all x, if x is a professor, then x smokes

2.4. The internal composition of a quantificational DP



- Clearly, we can substitute the noun 'professor' with any other noun and get a different quantificational statement:

(59) Every student smokes

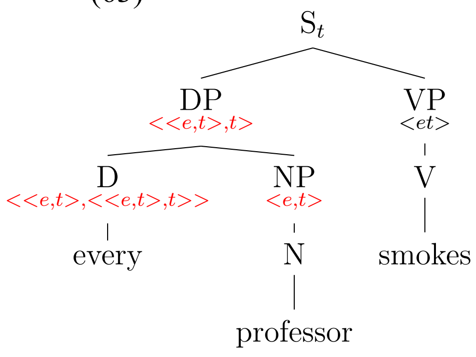
- We want to account for this fact
- Thus, we want to figure out what the semantics of the determiner 'every' is.

We call determiners of type <<e,t>,<<e,t>,t>> quantificational determiners.

Given that *every*, *no*, *some* are of type <<et>, <<e,t>, t>>, it follows that they compose with the NP via FA

(60) ⟦**every professor**⟧ = ⟦**every**⟧(⟦**professor**⟧)
(61) ⟦**no professor**⟧ = ⟦**no**⟧(⟦**professor**⟧)
(62) ⟦**a/some professor**⟧ = ⟦**a/some**⟧(⟦**professor**⟧)

(63)



Let's look at the result of putting together a determiner with a noun phrase again:

(64) ⟦**every**⟧(⟦**professor**⟧) = [$\lambda f_{<e,t>}$. for all x, if x is a **professor**, then f(x) = T]
(65) ⟦**no**⟧(⟦**professor**⟧) = [$\lambda f_{<e,t>}$. there is no **professor** x such that f(x) = T]
(66) ⟦**a/some**⟧(⟦**professor**⟧) = [$\lambda f_{<e,t>}$ .there is some **professor** x such that f(x) = T
]

Thus, the meaning of the determiners themselves is as follows:

(67) ⟦**every**⟧= [$\lambda p_{<e,t>}$. [$\lambda f_{<e,t>}$. for all x, if p(x)=T, then f(x) = T]]
(68) ⟦**no**⟧= [$\lambda p_{<e,t>}$. [$\lambda f_{<e,t>}$. there is no x such that p(x)=T and f(x) = T]]
(69) ⟦**a/some**⟧= [$\lambda p_{<e,t>}$. [$\lambda f_{<e,t>}$ .there is some x such that p(x)=T and f(x) = T]]

We call the argument denoted by the NP 'the domain of the quantifier'

We call the the argument denoted by the VP 'the scope of quantifier'

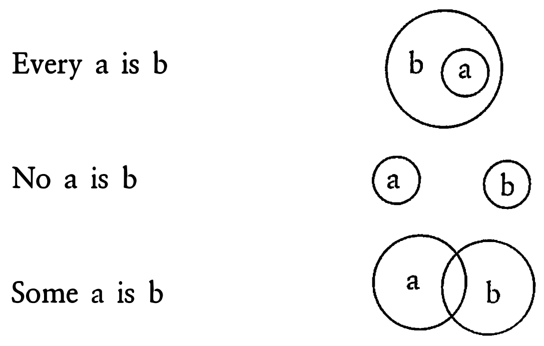
'Every professor smokes':
- 'Professor' is the domain of 'every'
- 'Smokes' is the scope of 'every'

(70)
⟦**S**⟧ = by FA
⟦**DP**⟧ (⟦**VP**⟧)= by FA
⟦**D**⟧ (⟦**NP**⟧) (⟦**VP**⟧)= by 5 applications of NN
⟦**every**⟧ **(**⟦**professor**⟧) (⟦**smokes**⟧)= by TN and the lexicon
[$\lambda g_{<et>}$.$\lambda f_{<et>}$. for all x, if g(x)=1, then f(x) = T] (λz_e.z is a professor) (λy_e.y smokes) =
$\lambda f_{<et>}$. for all x, if [λz_e.z is a professor](x)=T , then f(x) = T =
$\lambda f_{<et>}$. for all x, if x is a professor, then f(x) = T =
T iff for all x, if x is a professor, then [λy_e.y smokes](x) =T =
T iff for all x, if x is a professor, then x smokes

2.5 Quantifier meaning in terms of relation between sets

- We always said there is a close connection between functions and sets.

- One useful way of thinking about the meaning of a quantificational determiner is in terms of relation between the two sets picked by the characteristic function denoted by the restrictor and the scope.



(71) $[\![\textbf{every}]\!] = [\ \lambda g_{<e,t>} .\ [\ \lambda f_{<e,t>} .\ \{x: g(x)=T\} \subseteq \{y: f(y)= T\}]\ ]$
(72) $[\![\textbf{no}]\!] = [\ \lambda g_{<e,t>} .[\ \lambda f_{<e,t>} .\ \{x: g(x)= T\} \cap \{y: f(y)= T\}= \varnothing]\ ]$
(73) $[\![\textbf{a/some}]\!] = [\ \lambda g_{<e,t>} .\ [\ \lambda f_{<e,t>} .\ \{x: g(x)= T\} \cap \{y: f(y)= T\} \neq \varnothing]\ ]$

The two formulation are completely equivalent

(74) $[\![\textbf{every professor smokes}]\!]$ = T iff {x: x is a professor} $\subseteq$ {y: y smokes}
(75) $[\![\textbf{no professor smokes}]\!]$ = T iff {x: x is a professor} $\cap$ {y: y smokes} = $\varnothing$
(76) $[\![\textbf{a/some professor smokes}]\!]$ = T iff {x: x is a professor} $\cap$ {y: y smokes} $\neq$ $\varnothing$

2.6. Explaining the behavior of quantifiers

Argument 1

Earlier Observation:

Contrary to the predictions of a *type e* analysis, the following can hold:
- $[\![\textbf{no professor smokes Marlboros}]\!]$ = T and $[\![\textbf{no professor smokes}]\!]$ = F

Now we can explain this:
- $[\![\textbf{no professor smokes Marlboros}]\!]$ = T iff {x: x is a professor} $\cap$ {y: y smokes Marlboros} = $\varnothing$
- $[\![\textbf{no professor smokes}]\!]$ = F iff {x: x is a professor} $\cap$ {y: y smokes} $\neq$ $\varnothing$

Since the set of smokers can be bigger than the set of Marlboros smokers, it can be that the intersection of professors and Marlboros smokers is empty, but some professors smoke (in other words, the intersection of professors and smokers is not empty)

Argument 2

Earlier Observation:

Contrary to the predictions of a *type e* analysis, the following can hold:

- ⟦**a/some professor smokes** ⟧ = T *and* ⟦**a/some professor doesn't smoke** ⟧ = T

Now we can explain this:

- ⟦ **some professor smokes** ⟧ = T iff {x: x is a professor} ∩ {y: y smokes} ≠ ∅
- ⟦ **some professor doesn't smoke** ⟧ = T iff {x: x is a professor} ∩ {y: y does not smoke} ≠ ∅

If the set of professors include more than one person, then it is possible that some smoke and some do not.

Argument 3

Earlier Observation:
Contrary to the predictions of a *type e* analysis, the following can hold:
⟦ [**Every professor smokes**] or [**every professor doesn't smoke**] ⟧ = F

Now we can explain this:

- ⟦**every professor smokes** ⟧ = T iff {x: x is a professor} ⊆ {y: y smokes}
- ⟦**every professor doesn't smoke** ⟧ = T iff {x: x is a professor} ⊆ {y: y does not smoke}

It can be the case that both of those are false, because the sets of professors and smokers can have members in common, while it also being the case that not all professors are smokers

Argument 4

- Earlier observation: (77) is ambiguous:

(77) Some student saw every professor.

- Syntactic reorganization has a semantic effect of disambiguation:

(78) Some student is such that he saw every professor.
This requires that one student, say, John saw every professor.

(79) Every professor is such that some student saw her.
This could be true if for every professor there is a different student who saw her.

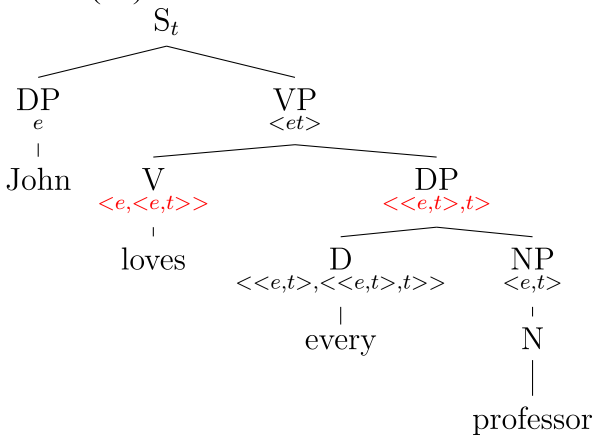
We need to develop some tools to understand this interaction!

3. Quantifiers in the object position

The problem: With the rules we have we cannot interpret quantifiers in the object position due to the type-mismatch:

- The quantificational DP is looking to combine with a predicate of individuals (something of type <e,t>)
- The V is of type <<e,t>,t>

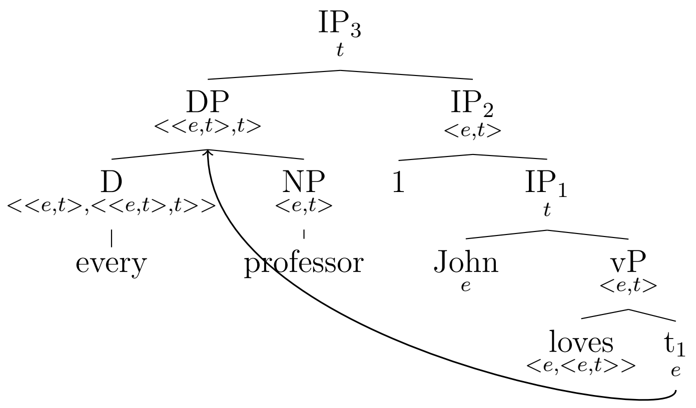
(80)



We are going to create the predicate of the right semantic type in syntax by:

- Moving 'every girl'
- Inserting a pronoun like expression of type e in its place. We are going to call this expression 'a trace'. It carries a numerical index
- Inserting a numerical index matching the index on the trace right below the DP
- Introducing a special rule that allows us to interpret the structure of the form [1 α] as a predicate of individuals.

(81)



3.1 Step 1: traces and pronouns

How do we interpret t_1 ?

- Some expression in a language do not have a fixed meaning.
- Their meaning can change depending on the context
- We call such expression variables
- For example, we cannot interpret (82) without some contexts.
- This means that the meaning of 'he' is not in the lexicon.

(82) He came.

- One sentence can have multiple variables referring to different individuals.
- In order to distinguish between them we are going to assume that they carry a numerical index.

(83) He_1 introduced him_2 to him_3.

- We are going to have a special function that takes care of the meaning of variables.
- This function maps a numerical index to an individual.
- We are going to call it 'the assignment function'
- For each conversation we can have a different assignment function
- This accounts for the fact that the meaning of variables is not fixed across the language.

(84) He_1 introduced him_2 to him_3.

(85)
$$g := \begin{bmatrix} 1 \rightarrow John \\ 2 \rightarrow Seth \\ 3 \rightarrow Mark \\ 4 \rightarrow Bill \end{bmatrix}$$

An alternative way of writing the same (representing g in terms of ordered pairs):

(86) g:={<1, John>, <2, Seth>, <3, Mark>,<4, Bill>}

(87) He_1 introduced him_2 to him_3.

- We need to make our interpretation function relativized to the assignment function.
- We are going to represent it as a superscript on the interpretation function.

$[\![\alpha]\!]^g$

- We are going to have a rule that tells us when to look at g

(88) **Trace and Pronoun Rule**
If α_n is a trace or a pronoun, n is a numerical index, g is a variable assignment and $n \in Dom(g)$, then $[\![\alpha_n]\!]^g = g(n)$

- We are going to assume that the interptetation function is always relativised to an assignment function.

- This assignment becomes relevant when an expression is a pronoun or a trace.

(89) $[\![he_1]\!]^g = g(1)$=John
(90) $[\![he_2]\!]^g = g(2)$=Seth

- Why? Because of the special rule we have for them!

(91) $[\![he_1]\!]^a = a(1)$=Seth
(92) $[\![he_2]\!]^a = a(2)$=Mark

(93)
$$a := \begin{bmatrix} 1 \rightarrow Seth \\ 2 \rightarrow Mark \\ 3 \rightarrow Bill \\ 4 \rightarrow John \end{bmatrix}$$

Note that:
(94) $[\![John]\!]^g = [\![John]\!]^a$=John
(95) $[\![smokes]\!]^g = [\![smokes]\!]^a = \lambda x_e$. x smokes

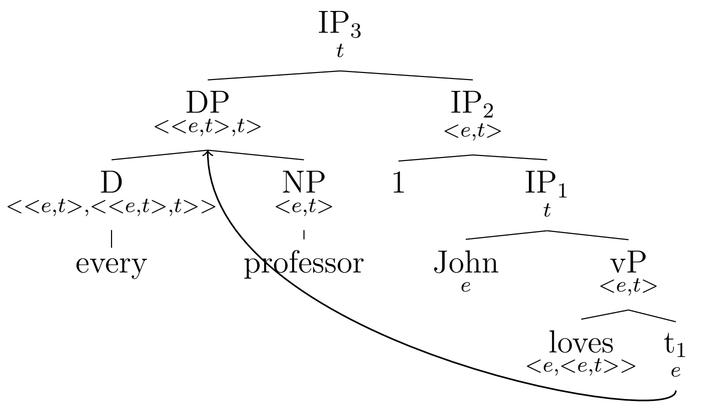
- Now we can give a more precise definition to the notion of a variable.

A terminal symbol α is a **variable** iff there are assignments g and a sich that $[\![\alpha]\!]^g \neq [\![\alpha]\!]^a$

- ‘He_1’ is a variable: it can change its denotation depending on the assignment function;
- ‘John’ is not a variable: it does not change its denotation depending on the assignment function

With the Step 1 we are ready to interpret one part of the tree, namely IP_1:

(96)



(97)

$[\![\mathbf{IP_1}]\!]^g$ = by FA
$[\![\mathbf{vP}]\!]^g([\![\mathbf{John}]\!]^g)$ = by FA
$[\![\mathbf{vP}]\!]^g([\![\mathbf{t_1}]\!]^g)([\![\mathbf{John}]\!]^g)$ = by TN
$\lambda x_e.\lambda y_e.$ y loves x $([\![\mathbf{t_1}]\!]^g)$ (**John**) = by T&P
$[\lambda x_e.[\lambda y_e.$ y loves x]] (g(1)) (**John**) =
T iff John loves g(1)

As you see, IP_1 is of type t, so we are not done constructing the right argument for 'every professor'!

The actual truth conditions we want:

$[\![(96)]\!]^g$ = T iff for all x: if x is a professor, then John loves x.

The meaning of 'every professor':

(98) $[\![\textbf{every professor}]\!]^g = [\lambda f.\ f \in D_{<e,t>}$: for all x, if x is a professor, then f(x) = T]

Thus, the desired argument of 'every professor':

(99) λx_e. John loves x

3.2 Predicate abstraction

(100) **Predicate abstraction**

If α is a branching node and $\{\beta\ \gamma\}$ is the set of its daughters, where β is a numerical index n, then for any variable assignment g, $[\![\alpha]\!]^g = \lambda x.\ [\![\gamma]\!]^{g(x/n)}$, where g(x/n) is a function that is just like g, but it assigns the value x to the numerical index n.

- If you have a structure of the form [a numerical index γ], then
- Write $\lambda x.$ before the $[\![\,]\!]^g$ and put the **sister** of the numerical index (like 1) into these $[\![\textbf{sister}]\!]^g$
- Make sure that the trace with the same numerical index is interpreted as x

- Do this by chaning the assignment function

- What is this?
  g[x/n]
- g is the assignment function
- g[x/n] is a modified assignment function such that:
  - n is in its domain
  - g[x/n](n) = x
  - for all m≠n, g(m) = g[x/n](n)

Possibility 1: the numerical index was not in the domain of g, g[x/1] differs from g in that it has 1 in its domain and it maps it to x

(101)

$$g := \begin{bmatrix} 2 \rightarrow Seth \\ 3 \rightarrow Mark \\ 4 \rightarrow Bill \end{bmatrix}$$

(102)

$$g[x/1] := \begin{bmatrix} 1 \rightarrow x \\ 2 \rightarrow Seth \\ 3 \rightarrow Mark \\ 4 \rightarrow Bill \end{bmatrix}$$

Possibility 2: g already had 1 in its domain and it was mapped to John. g[John/1] does not differ from g at all.

(103)

$$g := \begin{bmatrix} 1 \rightarrow John \\ 2 \rightarrow Seth \\ 3 \rightarrow Mark \\ 4 \rightarrow Bill \end{bmatrix}$$

(104)

$$g[John/1] := \begin{bmatrix} 1 \rightarrow John \\ 2 \rightarrow Seth \\ 3 \rightarrow Mark \\ 4 \rightarrow Bill \end{bmatrix}$$

Possibility 3: 1 was in the domain of g, but it was mapped to Mark. g[x/1] differs from g in that 1 is mapped to x

(105)

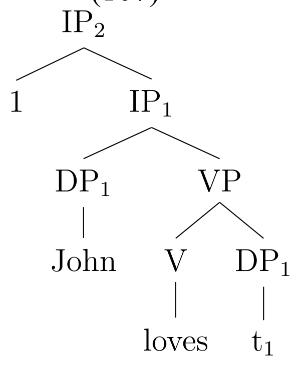
$$g := \begin{bmatrix} 1 \to Mark \\ 2 \to Seth \\ 3 \to Mark \\ 4 \to Bill \end{bmatrix}$$

(106)

$$g[x/1] := \begin{bmatrix} 1 \to x \\ 2 \to Seth \\ 3 \to Mark \\ 4 \to Bill \end{bmatrix}$$

- Given this rule, structures like IP_2 will always be interpreted as functions from individuals to whatever the type of IP_1 is.
- In this case IP_1 is of type t, as we have already established.
- Thus, IP_2 will be a predicate of individuals, in other words, an expression of type <e,t>

(107)



(108) $⟦IP_2⟧^g$ = by Predicate abstraction

λx. $⟦IP_1⟧^{g[x/1]}$ = by FA

λx. $⟦vP⟧^{g[x/1]}$ (⟦**John**⟧$^{g[x/1]}$) = by FA

λx. ⟦**loves**⟧$^{g[x/1]}$ ($⟦t_1⟧^{g[x/1]}$) (⟦**John**⟧$^{g[x/1]}$) = by TN, T&P

[λx. [λz. λy. y loves z] (g[x/1](1)) (John)]= by g[x/1]

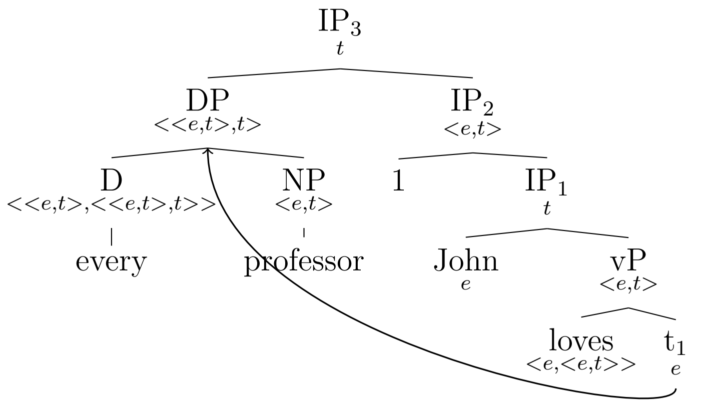
[λx. [λz. λy. y loves z] (x)(John)]

λx. John loves x

Binding

- We call the numerical abstractors 'binders'
- As you can see from the PA rule, their role is to remove the assignment dependency
- Roughly, 'variable binding' is any semantic operation which removes (or reduces) assignment dependency.

(109)



(110)

⟦IP$_2$⟧^{g} = λx. John loves x

⟦**every professor**⟧ = [λf. f∈D$_{<et>}$: for all z, if z is a professor, then f(z) = T]

⟦IP$_3$⟧^{g} = by FA

⟦DP⟧^{g} (⟦IP$_2$⟧^{g})= by our earlier computation

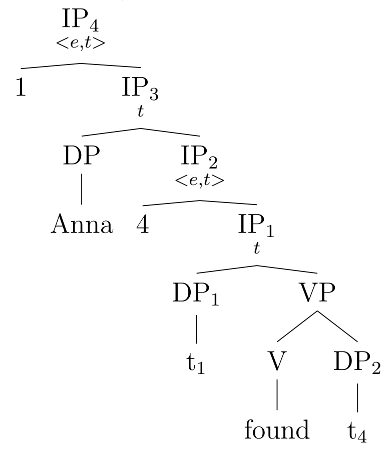
[λf. f∈D$_{<,>}$: for all z, if z is a professor, then f(z) = T] (λx. John loves x) =

T iff for all z, if z is a professor, then [λx. John loves x](z) = T =

T iff for all z, if z is a professor, then John loves z

- Nothing prevents us from having multiple abstractions in one tree

(111)



(112)

⟦**IP$_4$**⟧^{g} = by PA

λx. ⟦**IP$_3$**⟧$^{g[x/1]}$= by FA

λx. ⟦**IP$_2$**⟧$^{g[x/1]}$ (⟦**Anna**⟧$^{g[x/1]}$) = by PA

[λx. [λz. ⟦**IP$_1$**⟧$^{g[x/1,\ z/4]}$](⟦**Anna**⟧$^{g[x/1]}$)] = by TN and lexicon

$[\lambda x.\ [\lambda z.\ [\![\mathbf{IP_1}]\!]^{g[x/1,\ z/4]}\ ](\text{Anna})] =$
$\lambda x.\ \ [\![\mathbf{IP_1}]\!]^{g[x/1,\ \text{Anna}/4]}$

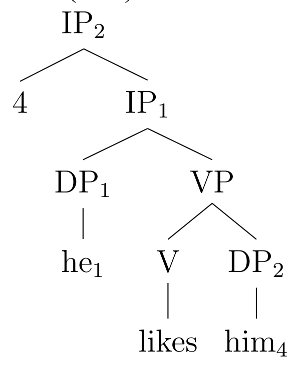
$[\lambda x.[\![\mathbf{IP_1}]\!]^{g[x/1,\ \text{Anna}/4]}]$=by 2 application of FA
$[\lambda x.\ [\lambda y.[\lambda a.\ \text{a found y}]]\ ([\![\mathbf{t_4}]\!]^{g[x/1,\ \text{Anna}/4]})\ ([\![\mathbf{t_1}]\!]^{g[x/1,\ \text{Anna}/4]})\ ]$= by T&P
$[\lambda x.\ [\lambda y.\ [\lambda a.\ \text{a found y}\ ]]\ (g[x/1, \text{Anna}/4]\ (4))\ (g[x/1, \text{Anna}/4](1))] =$ by $g[x/1, \text{Anna}/4]$
$[\lambda x.\ [\lambda y.\ [\lambda a.\ \text{a found y}\ ]](\text{Anna})\ (x)] =$
$[\lambda x.\ \text{x found Anna}]$

Practice:

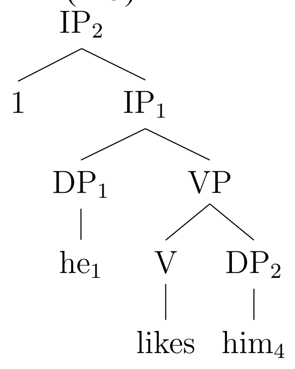
(113)

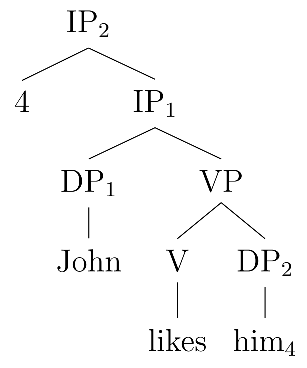
$$g := \begin{bmatrix} 1 \to John \\ 2 \to Seth \\ 3 \to Mark \\ 4 \to Bill \end{bmatrix}$$

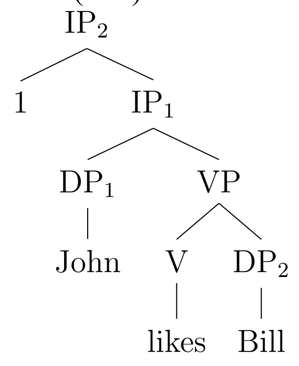
(114)



(115)



(116)


(117)


4. Modeling quantifier scope

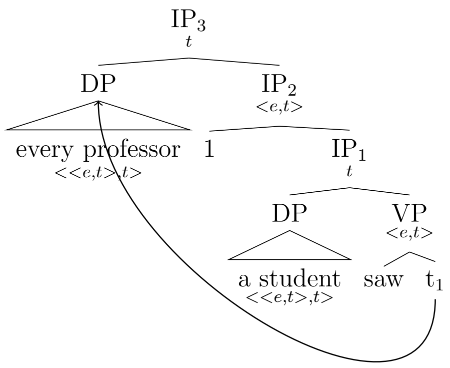
(118) A student saw every professor.

Reading 1: every professor was seen by a (possibly different) student
This reading is called 'the inverse scope reading'

Reading 2: one student saw all professors

Reading 1:

(119)

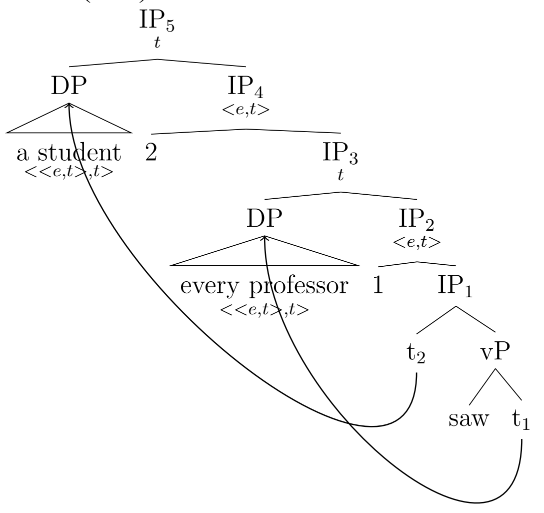


⟦**IP$_2$**⟧^{g} = λy. there is an x such that: x is a student and x saw y

⟦**IP$_3$**⟧^{g} = T iff for every y: if y is a professor, then there is an x such that: x is a student and x saw y

Reading 2: one student saw all professors

(120)



(121) ⟦**IP$_4$**⟧^{g} = λy. for every x: if x is a professor, then y saw x

(122) ⟦**IP$_5$**⟧^{g} = T iff there is y such that: y is a student and for every x: if x is a professor, then y saw x

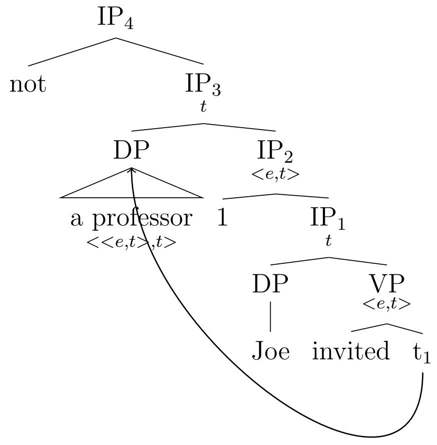
(123) Joe didn't invite a professor.

Reading 1: Joe did not invite any professor, not even one

Reading 2: There is one specific professor such that John did not invite her. (This reading is compatible with Joe inviting other professors)

Reading 1:

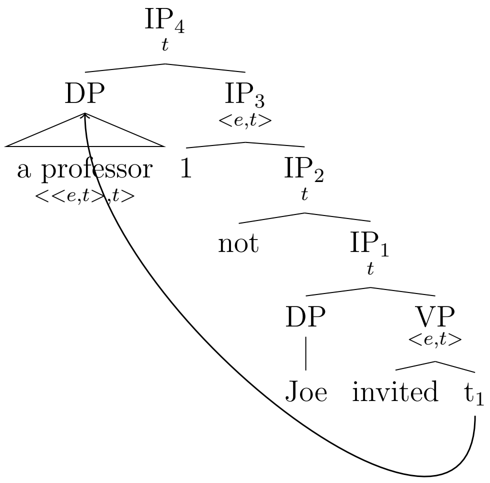
(124)



⟦**IP$_4$**⟧^{g}= T iff it is not the case that there is an x such that: x is a professor and Joe invited x

Reading 2:

(125)



(126)

⟦**IP$_3$**⟧^{g}= λy. it is not the case that Joe invited y

⟦**IP$_4$**⟧^{g}= there is a y such that: y is a professor and it is not the case that Joe invited y

5. Quantifier raising?

(127) Some student read every book on the list

Scenario: There are 3 students John, Bill and Mary.
John read book A, Bill book B, Mary book C.
There is no individual student who read every book, but every book was read by one student or another

We represented QR as a covert (silent) movement operation.

- In order to be able to give ‘every book on the list’ a scope over ‘some student’, we moved it to a higher position.

- The test for 'every'>'some' reading is if the sentence is judged as true if for each book on the list there could be a different student who read it.

- We represented QR as a movement operation.

- One thing we know from the syntactic literature is that movement is sensitive to various boundaries (called 'islands').

- Thus, the prediction of the movement theory of quantifier scope is that there will be some contexts where the inverse scope is unavailable.

5.1 Empirical argument one: a finite clause

(128) Some student said that every professor is fantastic.

- Some >every
  Scenario: John said: 'every professor is fantastic!'

- *Every>some
  Scenario: There are 3 students John, Bill and Mary.
  John said professor A is fantastic, Bill said professor B is fantastic, Mary said professor C fantastic.
  There is no individual student who likes every professor, but for every professor there is a student who said that that professor is fantastic.

According to the movement theory of scopal interaction, to get the inverse scope reading 'every professor' would have to move at LF and be higher than 'some student'.

The movement is impossible from this position:

(129) *Who did some student say that _ is fantastic?

This is why this reading is not available!

5.1 Empirical argument two: a relative clause

(130) John read a book that was written by every author in the list.

- Some >every

Scenario: John read a book. This one specific book was written by every author on the list.

- *Every>some

For every author on the list, John read a book written by that author

The movement is impossible from this position:

(131) *Who did John read a book that was written by __.

7. Other variable binding contexts

- We saw how traces can get bound and serve to form the predicates that we need for our derivations.
- But notice that pronouns are interpreted in exactly the same way as traces.
- Then we expect them to serve the same function as traces: they can be assignment independent and serve to form predicates
- Do we have such cases?

The sentence in (132) is ambiguous.

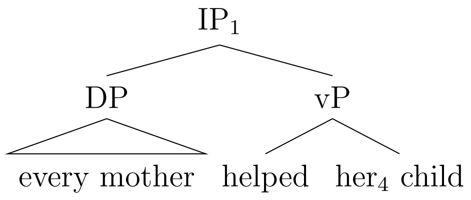
Bound readings of pronouns

(132) Every mother helped her child.

- Reading 1: every mother helped a child of a particular person I am pointing at (say, Mary).
- Reading 2: Every mother helped her own child.

Reading 1: We say that in this case the pronoun is 'free'. This means it depends on the specific assignment function we picked for this context

(133)

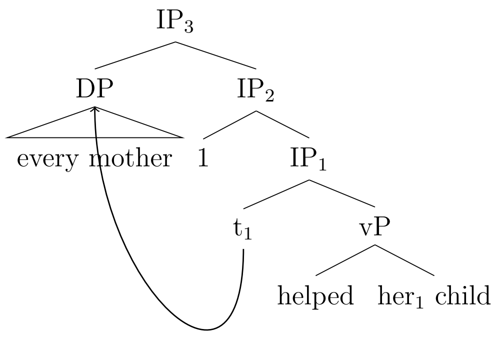


(134) $[\![$**vP**$]\!]^g$= λy. y helped the child of g(4)
(135) $[\![$**IP$_1$**$]\!]^g$= T iff for all y: if y is a mother, then y helped the child of g(4)
(136) $[\![$**IP$_1$**$]\!]^g$= T iff for all y: if y is a mother, then y helped the child of Mary

Reading 2: the pronoun is bound

- In this reading there is no specific person 'her' refers to. The value of 'her' varies with mothers.
- 'Every' scrolls through the individuals of the world and checks if it is true that if an individual is a mother, then she helped the child of that individual.
- We call this reading 'a bound reading'
- 'Her' does not depend on the specific assignment function we assume in the context.

(137)



(138) ⟦**IP**$_2$⟧^{g}= λx.x helped x's child

(139) ⟦**IP**$_3$⟧^{g}= T iff for all y: if y is a mother, then y helped the child of y