Semantics Bootcamp Part I: Fundamentals

Elizabeth Coppock

NASSLLI 2012, Austin Texas

1 Semantics

Semantics: The study of meaning. What is **meaning**? How can you tell whether somebody or something **understands**? Does Google understand language?

An argument that it does not: Google can't do **inferences**:

(1) Everyone who smokes gets cancer → No heavy smoker avoids cancer

(cf. "No alleged smoker avoids cancer"...)

A hallmark of a system or agent that understands language / grasps meaning is that it can do these kinds of inferences.

Another way to put it: A good theory of meaning should be able to account for the conditions under which one sentence implies another sentence.

Some different kinds of inferences:

1. **Entailment** (domain of semantics): A entails B if and only if whenever A is true, B is true too. E.g. *Obama was born in 1961* entails *Obama was born in the 1960s*.
2. **Presupposition** (semantics/pragmatics): A presupposes B if and only if an utterance of A takes B for granted. E.g. *Sue has finally stopped smoking* presupposes that Sue smoked in the recent past.

3. **Conversational Implicature** (pragmatics): A conversationally implicates B if and only if a hearer can infer B from an utterance of A by making use of the assumption that the speaker is being cooperative. E.g. *Some of the students passed* conversationally implicates that not all of the students passed.

The primary responsibility for a theory of *semantics* is to account for the conditions under which one sentence *entails* another sentence.

Other nice things a theory of semantics could do: Account for presuppositions, contradictions, equivalences, semantic ill-formedness, distribution patterns.

Strategy: Assign **truth conditions** to sentences. The truth conditions are the conditions under which the sentence is true. Knowing the meaning of a sentence does not require knowing whether the sentence is in fact true; it only requires being able to discriminate between situations in which the sentence is true and situations in which the sentence is false.

(Cf. Heim & Kratzer's bold first sentence: "To know the meaning of a sentence is to know its truth conditions.")

The strategy of assigning truth conditions will allow us to account for entailments. If the circumstances under which A is true include the circumstances under which B is true, then A entails B.

How to assign truth conditions to sentences of natural languages like English? Montague's idea: Let's pretend that English is a formal language.

> I reject the contention that an important theoretical difference exists between formal and natural languages. ... In the present paper I shall accordingly present a precise treatment, culminating in a theory of truth, of a formal language that I believe may reasonably be regarded as a fragment of ordinary English. ... The treatment given here will be found to resemble the usual syntax and model theory (or semantics) [due to Tarski] of the predicate calculus, but leans rather heavily on the intuitive aspects of certain recent developments in intensional logic [due to Montague himself]. (Montague 1970b, p.188 in Montague 1974)

2 Predicate calculus

Predicate calculus is a logic. Logics are formal languages, and as such they have syntax and semantics.

Syntax: specifies which expressions of the logic are well-formed, and what their syntactic categories are.

Semantics: specifies which objects the expression correspond to, and what their semantic categories are.

2.1 Syntax of Predicate Calculus

2.1.1 Atomic symbols

Formulas are built up from atomic symbols drawn from the following syntactic categories:

- **individual constants**: JOHN, MARY, TEXAS, 4...
- **variables**: $x, y, z, x', y', z', ...$
- **predicate constants**
  - **unary predicate constants**: EVEN, ODD, SLEEPY,..
  - **binary predicate constants**: LOVE, OWN, >, ...
  - ...
- **function constants**:
  - **unary function constants**: MOTHER, ABSOLUTE_VALUE, ...
  - **binary function constants**: DISTANCE, +, −,...
  - ...
- **logical connectives**: $\wedge, \vee, \rightarrow, \leftrightarrow, \neg$
- **quantifiers**: $\forall, \exists$

I write constants in SMALL CAPS and variables in $italics$. But I will ignore quantifiers and variables today.

2.2 Syntactic composition rules

How to build complex expressions (metalanguage variables are Greek letters):

- If π is a n-ary predicate and $\alpha_1 ... \alpha_n$ are terms, then $\pi(\alpha_1,...\alpha_n)$ is an atomic formula.
  - If π is a unary predicate and α is a term, then $\pi(\alpha)$ is an atomic formula.
  - If π is a binary predicate and α_1 and α_2 are terms, then $\pi(\alpha_1, \alpha_2)$ is an atomic formula.
- If $\alpha_1 ... \alpha_n$ are terms, and γ is a function constant with arity n, then $\gamma(\alpha_1, ..., \alpha_n)$ is a term.
- If ϕ is a formula, then $\neg\phi$ is a formula.
- If ϕ is a formula and ψ is a formula, then $[\phi \wedge \psi]$ is a formula, and so are $[\phi \vee \psi]$, $[\phi \rightarrow \psi]$, and $[\phi \leftrightarrow \psi]$.

Expression	Syntactic category
JOHN,MARY	(individual) constant
x	variable
HAPPY, EVEN	unary predicate constant
LOVE, >	binary predicate constant
LOVE(JOHN, MARY)	(atomic) formula
HAPPY(x)	(atomic) formula
$x > 1$	(atomic) formula
MOTHER	unary function constant
MOTHER(JOHN)	term

2.3 Semantics of Predicate Calculus

Each expression belongs to a certain semantic type. The types of our predicate calculus are: **individuals**, **sets, relations**, **functions,** and **truth values**.

2.3.1 Sets

Set. An abstract collection of distinct objects which are called the *members* or *elements* of that set. Elements may be concrete (like the beige 1992 Toyota Corolla

I sold in 2008, David Beaver, or your computer) or abstract (like the number 2, the English phoneme /p/, or the set of all Swedish soccer players). The elements of a set are not ordered, and there may be infinitely many of them or none at all.

You can specify the members of a set in two ways:

1. By listing the elements, e.g.:
   $\{\text{Marge}, \text{Homer}, \text{Bart}, \text{Lisa}, \text{Maggie}\}$
2. By description, e.g: $\{x | x \text{ is a human member of the Simpsons family}\}$

Element. We write 'is a member of' with $\in$.

Empty set. The *empty set,* written $\varnothing$ or $\{\}$, is the set containing no elements.

Subset. A is a *subset* of B, written $A \subseteq B$, if and only if every member of A is a member of B.

$$A \subseteq B \text{ iff for all } x\text{: if } x \in A \text{ then } x \in B.$$

Proper subset. A is a *proper subset* of B, written $A \subset B$, if and only if A is a subset of B and A is not equal to B.

$$A \subset B \text{ iff (i) for all } x\text{: if } x \in A \text{ then } x \in B \text{ and (ii) } A \neq B.$$

Powerset. The *powerset* of A, written $\wp(A)$, is the set of all subsets of A.

$$\wp(A) = \{S | S \subseteq A\}$$

Set Union. The *union* of A and B, written $A \cup B$, is the set of all entities x such that x is a member of A or x is a member of B.

$$A \cup B = \{x | x \in A \text{ or } x \in B\}$$

Set Intersection. The *intersection* of A and B, written $A \cap B$, is the set of all entities x such that x is a member of A and x is a member of B.

$$A \cap B = \{x | x \in A \text{ and } x \in B\}$$

2.3.2 Ordered pairs and relations

Ordered pair. Sets are not ordered.

$$\{\text{Bart}, \text{Lisa}\} = \{\text{Lisa}, \text{Bart}\}$$

But the elements of an *ordered pair* written $\langle a, b\rangle$ are ordered. Here, a is the *first member* and b is the *second member*.

$$\langle \text{Bart}, \text{Lisa}\rangle \neq \langle \text{Lisa}, \text{Bart}\rangle$$

We can also have ordered triples.

$$\{\text{Bart}, \text{Lisa}, \text{Maggie}\} = \{\text{Maggie}, \text{Lisa}, \text{Bart}\}$$

$$\langle \text{Bart}, \text{Lisa}, \text{Maggie}\rangle \neq \langle \text{Maggie}, \text{Lisa}, \text{Bart}\rangle$$

Relation. A (binary) relation is a set of ordered pairs. For example, the 'older-than' relation among Simpsons kids:

$$\{\langle \text{Bart}, \text{Lisa}\rangle, \langle \text{Lisa}, \text{Maggie}\rangle, \langle \text{Bart}, \text{Maggie}\rangle\}$$

Note that this is a *set*. How many elements does it have?

Domain. The *domain* of a relation is the set of entities that are the first member of some ordered pair in the relation.

Range. The *range* of a relation is the set of entities that are the second member of some ordered pair in the relation.

2.3.3 Functions

Function. A function is a special kind of relation. A relation R from A to B is a function if and only if it meets both of the following conditions:

1. Each element in the domain is paired with just one element in the range.
2. The domain of R is equal to A

A function gives a single **output** for a given **input**.

Are these relations functions?

$$f_1 = \{\langle \text{Bart}, \text{Lisa}\rangle, \langle \text{Lisa}, \text{Maggie}\rangle, \langle \text{Bart}, \text{Maggie}\rangle\}$$

$$f_2 = \{\langle \text{Bart}, \text{Lisa}\rangle, \langle \text{Lisa}, \text{Maggie}\rangle\}$$

Easier to see when you notate them like this:

$$f_1 = \begin{bmatrix} \text{Bart} & \rightarrow & \text{Lisa} \\ & \rightarrow & \text{Maggie} \\ \text{Lisa} & \rightarrow & \text{Maggie} \end{bmatrix}$$

$$f_2 = \begin{bmatrix} \text{Bart} & \rightarrow & \text{Lisa} \\ \text{Lisa} & \rightarrow & \text{Maggie} \end{bmatrix}$$

Lambda (λ) notation. Just as sets can be specified either by listing the elements or by description, functions can be described either by listing the ordered pairs that are members of the relation or by description. To describe a function, the symbol λ is used.

λ-terms usually follow the following schema:

$$\lambda\alpha\ .\ \gamma$$

where

- α is the argument variable (a letter that stands for an arbitrary argument of the function we are defining)
- γ is the value description (specifies the value that our function assigns to α)

Example:

$$\lambda x\ .\ x + 1$$

- x is the argument variable
- $x + 1$ is the value description

Function application. $F(a)$ denotes 'the result of applying function F to argument a' or F of a' or 'F applied to a'. If F is a function that contains the ordered pair $\langle a, b\rangle$, then:

$$F(a) = b$$

This means that given a as input, F gives b as output.

The result of applying a function specified using λ notation to its argument can normally be written as the value description part (γ), with the argument substituted for the argument variable (α).

$$[\lambda x\ .\ x + 1](4) = 4 + 1 = 5$$

2.3.4 Semantic Types

Basic types:

- e is the type of individuals; all individuals have type e
- t is the type of truth values; the truth values 0 and 1 have type t

Complex types:

- If σ and τ are semantic types, then $\sigma \times \tau$ is the type of an ordered pair whose first element is of type σ and whose second element is of type τ.
- If σ is a type, then (σ) is the type of sets containing elements of type σ
- If $\sigma_1, \sigma_2, ...$ are types, then $(\sigma_1, \sigma_2, ...)$ is the type of a relation consisting of ordered pairs of type $\sigma_1 \times \sigma_2 \times ...$
- If σ and τ are semantic types, then $\langle \sigma, \tau \rangle$ is the semantic type of functions whose domain is the set of entities of type σ and whose range is the set of entities of type τ.
- Nothing else is a semantic type.

Examples:

- $\langle e, t\rangle$ is the type of functions from individuals to truth values
- $\langle\langle e, t\rangle, t\rangle$ is the type of functions from [functions from individuals to truth values] to truth values
- (e) is the type of sets of individuals
- $e \times e$ is the type of an ordered pair of individuals
- (e, e) is the type of a binary relation among individuals

Expression	Syntactic category	Semantic type
JOHN,MARY	(individual) constant	e
HAPPY, EVEN	unary predicate constant	(e)
LOVE, >	binary predicate constant	(e, e)
$\text{LOVE}(\text{JOHN}, \text{MARY})$	(atomic) formula	t
$\text{HAPPY}(x)$	(atomic) formula	t
$x > 1$	(atomic) formula	t
MOTHER	unary function constant	$\langle e, e\rangle$
$\text{MOTHER}(\text{JOHN})$	term	e

Domains. If σ is a type, then D_σ is the set of things with that type. For example:

- $D_t = \{0, 1\}$
- $D_{\langle e,t\rangle} = \{f : f$ is a function from D_e to $D_t\ \}$
- $D_{\langle e,\langle e,t\rangle\rangle} = \{f : f$ is a function from $D_{\langle e,t\rangle}$ to $D_t\ \}$

2.3.5 Set theory as meta-language

All these set-theoretic symbols are formal symbols, but they are not part of the language for which we are giving a semantics. They are being used to characterize the *values* that expressions of predicate calculus will have.

In that sense, we are using the language of set theory as a meta-language.

2.3.6 Models and interpretation functions

Interpretation with respect to a model. Expressions of predicate calculus are *interpreted* in *models*. Models consist of a domain of individuals D and an interpretation function I which assigns values to all the constants:

$$\mathbf{M} = \langle D, I\rangle$$

An interpretation function $[\![\,]\!]^M$, built up recursively on the basis of the basic interpreation function I, assigns to every expression α of the language (not just the constants) a **semantic value** $[\![\alpha]\!]^M$.

Here are two models, M_r and M_f (r for "real", and f for "fantasy"/"fiction"/"fake"):

$$M_r = \langle D, I_r\rangle$$

$$M_f = \langle D, I_f\rangle$$

They share the same domain:

$$D = \{\text{Maggie}, \text{Bart}, \text{Lisa}\}$$

In M_r, Bart is happy, but Maggie and Lisa are not:

$$I_r(\text{HAPPY}) = [\![\text{HAPPY}]\!]^{M_r} = \{\text{Bart}\}$$

In M_f, everybody is happy:

$$I_f(\text{HAPPY}) = [\![\text{HAPPY}]\!]^{M_f} = \{\text{Maggie}, \text{Bart}, \text{Lisa}\}$$

Both interpretations assign the constant MAGGIE to Maggie:

$$I_r(\text{MAGGIE}) = [\![\text{MAGGIE}]\!]^{M_r} = \text{Maggie}$$

$$I_f(\text{MAGGIE}) = [\![\text{MAGGIE}]\!]^{M_f} = \text{Maggie}$$

What is the interpretation of $\text{HAPPY}(\text{MAGGIE})$? It should come out as false in M_r, and true in M_f. So what we want to get is:

$$[\![\text{HAPPY}(\text{MAGGIE})]\!]^{M_r} = 0$$

$$[\![\text{HAPPY}(\text{MAGGIE})]\!]^{M_f} = 1$$

What tells us this?

2.3.7 Interpretation rules

- **Constants**
  If α is a constant, then $[\![\alpha]\!]^M$ = I(α).
- **Complex terms**
  If $\alpha_1 ... \alpha_n$ are terms and γ is a function constant with arity n, then $[\![\gamma(\alpha_1, ...\alpha_n)]\!]^M$ is $[\![\gamma]\!]^M([\![\alpha_1]\!]^M, ..., [\![\alpha_n]\!]^M)$.
- **Atomic formulae**
  If π is an n-ary predicate and $\alpha_1 ... \alpha_n$ are terms, $[\![\pi(\alpha_1, ..., \alpha_n)]\!]^M = 1$ iff
  $$\langle [\![\alpha_1]\!]^M, ..., [\![\alpha_n]\!]^M \rangle \in [\![\pi]\!]^M$$
  If π is a unary predicate and α is a term, then $[\![\pi(\alpha)]\!]^M = 1$ iff
  $$[\![\alpha]\!]^M \in [\![\pi]\!]^M$$
- **Negation**
  $[\![\neg\phi]\!]^M = 1$ if $[\![\phi]\!]^M = 0$; otherwise $[\![\neg\phi]\!]^M = 0$.
- **Connectives**
  $[\![\phi \wedge \psi]\!]^M = 1$ if $[\![\phi]\!]^M = 1$ and $[\![\psi]\!]^M = 1$; 0 otherise. Similarly for $[\![\phi \vee \psi]\!]^M$, $[\![\phi \rightarrow \psi]\!]^M$, and $[\![\phi \leftrightarrow \psi]\!]^M$.

Example. Because $\textsc{happy}$ is a unary predicate and $\textsc{Maggie}$ is a term, we can use the rule for atomic formulae to figure out the value of $\textsc{happy}(\textsc{Maggie})$.

$[\![\textsc{happy}(\textsc{Maggie})]\!]^{M_f} = 1$ iff $[\![\textsc{Maggie}]\!]^{M_f} \in [\![\textsc{happy}]\!]^{M_f}$,
i.e. iff Maggie $\in \{\text{Maggie}, \text{Bart}, \text{Lisa}\}$.

$[\![\textsc{happy}(\textsc{Maggie})]\!]^{M_r} = 1$ iff $[\![\textsc{Maggie}]\!]^{M_r} \in [\![\textsc{happy}]\!]^{M_r}$,
i.e. iff Maggie $\in \{\text{Bart}\}$.

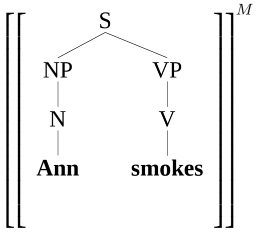
How about: $\neg\textsc{happy}(\textsc{Bart}) \vee \textsc{love}(\textsc{Bart}, \textsc{Maggie})$?

3 English as a formal language

Montague: “I reject the contention that an important theoretical difference exists between formal and natural languages.”

So we will put (parsed) English inside the denotation brackets, instead of logic.

Example:



= 1 iff Ann smokes (according to M).

(Wrong: $[\![\textbf{Ann smokes}]\!]^M$)

Following Heim & Kratzer, I use bold face for object language inside denotation brackets here, but often I am lazy about this.

Scientific question: What semantic composition rules do we need in order to calculate the values of complex expressions from the values of their parts?

Frege’s conjecture: All semantic composition is functional application.

3.1 A fragment of English (H&K 1998, ch. 2)

3.1.1 Inventory of denotations

(i) Elements of D_e, the set of actual individuals
(ii) Elements of {0,1}, the set of truth values
(iii) Functions from D_e to {0,1}

3.1.2 Lexicon

$[\![\textbf{Ann}]\!]^{M_0}$	=	Ann
$[\![\textbf{Jan}]\!]^{M_0}$	=	Jan
$[\![\textbf{works}]\!]^{M_0}$	=	$\lambda x : x \in D_e \,.\, x$ works
$[\![\textbf{smokes}]\!]^{M_0}$	=	$\lambda x : x \in D_e \,.\, x$ smokes

Heim & Kratzer's use of λ-notation:

Read "$[\lambda\alpha : \phi \,.\, \gamma]$" as either (i) or (ii), whichever makes sense.
(i) "the function which maps every α such that ϕ to γ"
(ii) "the function which maps every α such that ϕ to 1, if γ, and to 0 otherwise"

Notice that (lambdified) English is the *meta-language*, the language with which we describe the semantic values that object language expressions may have.

The expression 'λx : x $\in D_e$. x smokes' is supposed to describe a particular function, which we could specify more explicitly using set-theoretic notation. If Ann, Jan and Maria are the only individuals in the domain (so $D_e = \{$Ann, Jan, Maria$\}$), and Ann and Jan smoke and Maria does not, then:

$$[\![\textbf{smokes}]\!]^{M_0} = \{\langle \text{Ann}, 1\rangle, \langle \text{Jan}, 1\rangle, \langle \text{Maria}, 0\rangle\} = \begin{bmatrix} \text{Ann} & \to & 1 \\ \text{Jan} & \to & 1 \\ \text{Maria} & \to & 0 \end{bmatrix}$$

So there's two kinds of formal stuff:

- meta-language (λ, symbols of set theory)
- object language, when we're considering the syntax and semantics of a formal language (the symbols of predicate calculus)

3.1.3 Composition rules

Syntax-driven interpretation rules (boring)

(S1) If α has the form $\begin{array}{c}\text{S}\\ \diagup\ \ \diagdown\\ \beta\qquad\gamma\end{array}$ then $[\![\alpha]\!]^M = [\![\gamma]\!]^M([\![\beta]\!]^M)$.

(S2) If α has the form $\begin{array}{c}\text{NP}\\ |\\ \beta\end{array}$ then $[\![\alpha]\!]^M = [\![\beta]\!]^M$.

(S3) If α has the form $\begin{array}{c}\text{VP}\\ |\\ \beta\end{array}$ then $[\![\alpha]\!]^M = [\![\beta]\!]^M$.

(S4) If α has the form $\begin{array}{c}\text{N}\\ |\\ \beta\end{array}$ then $[\![\alpha]\!]^M = [\![\beta]\!]^M$.

(S5) If α has the form $\begin{array}{c}\text{V}\\ |\\ \beta\end{array}$ then $[\![\alpha]\!]^M = [\![\beta]\!]^M$.

Type-driven interpretation rules (cool)

(2) **Terminal Nodes** (TN)
If α is a terminal node, $[[\alpha]]^M$ is specified in the lexicon.

(3) **Non-Branching Nodes** (NN)
If α is a non-branching node, and β is its daughter node, then $[[\alpha]]^M = [[\beta]]^M$.

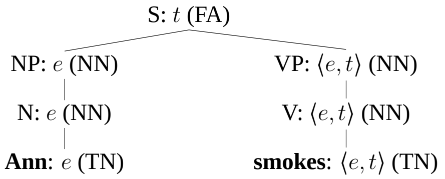
(4) **Functional Application** (FA)
If α is a branching node, $\{\beta, \gamma\}$ is the set of α's daughters, and $[[\beta]]^M$ is a function whose domain contains $[[\gamma]]^M$, then $[[\alpha]]^M = [[\beta]]^M([[\gamma]]^M)$.

Note: *Functional application* has two meanings!

1. The process of applying a function to an argument (a.k.a. "β-reduction")
2. The *composition rule* that allows us to compute the semantic value of a *phrase* given the semantic values of its parts.

Overview of how the compositional derivation of the truth conditions for *Ann smokes* will go:

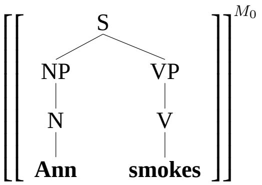
(5)

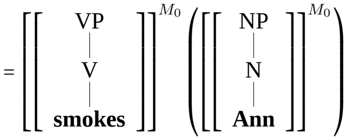

S: t (FA)
NP: e (NN)
VP: $\langle e,t\rangle$ (NN)
N: e (NN)
V: $\langle e,t\rangle$ (NN)
Ann: e (TN)
smokes: $\langle e,t\rangle$ (TN)

An alternative strategy would have been to treat *smokes* as a set of individuals, like we treated the predicate HAPPY in predicate calculus. But this would not have allowed us to use Functional Application. We use the **characteristic function** of that set instead.

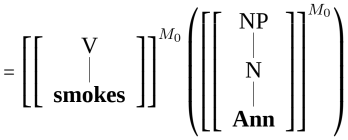
f is the **characteristic function** of a set S iff for all x in the relevant domain, $f(x) = 1$ if $x \in S$, and $f(x) = 0$ otherwise.

Compositional derivation of the truth conditions:





by Functional Application



by Non-branching Nodes

= $[\![$ **smokes** $]\!]^{M_0}\left([\![$ **Ann** $]\!]^{M_0}\right)$ by Non-branching Nodes (× 3)

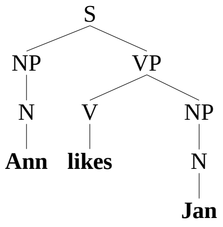
= $[\lambda x : x \in D_e \,.\, x$ smokes$]([\![$**Ann**$]\!]^{M_0})$ by Terminal Nodes

= $[\lambda x : x \in D_e \,.\, x$ smokes$]$(Ann) by Terminal Nodes

= 1 iff Ann smokes. (by β-reduction)

3.2 Transitive and ditransitive verbs

Semantics for transitive verbs. Example:



So that we can use FA for all the branching nodes, the VP should be a function from individuals to truth values (type $\langle e,t\rangle$).

So the semantic value of the transitive verb **likes** should be *a function from individuals to functions from individuals to truth values*, i.e, it should have the following type:

$$\langle e, \langle e,t\rangle\rangle$$

For example:

$$\begin{bmatrix} \text{Jacob} & \to & \begin{bmatrix} \text{Maria} & \to & 0 \\ \text{Jacob} & \to & 0 \end{bmatrix} \\ \text{Maria} & \to & \begin{bmatrix} \text{Maria} & \to & 1 \\ \text{Jacob} & \to & 1 \end{bmatrix} \end{bmatrix}$$

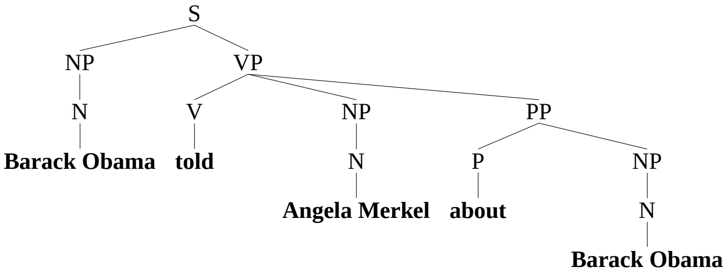
(6) $[\![$**likes**$]\!]$ = $\lambda x : x \in D.[\lambda y : y \in D \,.\, y$ likes $x]$

(7) $[\![$**likes**$]\!]([\![$**Jan**$]\!])$ = $[\lambda x : x \in D \,.\, [\lambda y : y \in D.y$ likes $x]]$(Jan)
= $\lambda y : y \in D \,.\, y$ likes Jan

(8) $[[\![$**likes**$]\!]([\![$**Jan**$]\!])][\![$**Ann**$]\!]$ = $[\lambda y : y \in D \,.\, y$ likes Jan](Ann)
= 1 iff Ann likes Jan

Ditransitive verbs. Suppose we have a phrase structure rule for ditransitive verbs like *tell*, that generates sentences like *X told Y about Z*. It could generate trees like this for example:

(9)



What type should *tell* be? What order should the arguments come in?