Semantik, Modul 1003

Composition rules, functional application, Schönfinkelization

Heim & Kratzer (1998), ch. 2-2.4

Leipzig University

April 11th, 2024

Fabian Heck
Folien by Imke Driemel

Recap from last session

1. the meaning of a sentence is the set of conditions that must be met in order for that sentence to be true
2. Frege introduces the difference between *Sinn* and *Bedeutung*, i.e. **intension** and **extension**, the latter is what we focus on in this class
3. the extension of many words is the thing they point to in the world
4. the extension of sentences is their truth conditions
5. sentence meaning is compositional:

(1) **Compositionality:**
The meaning of an expression is determined by the meaning of its component parts and the way in which they are combined and nothing else.

6. syntax is crucial in determining how components of meaning are combined
7. predicates are special kinds of functions whose range is restricted to 1 and O

Summary on sets

We said last week that we sometimes use set notation and sometimes function notation:

(2) ⟦sleep⟧:=

a. function-talk: *sleep(x)* = 1 iff *x sleeps*

b. set-talk: {x | x is a sleeper}

What are sets again?
A set is an abstract collection of objects. These can be real-world objects, concepts, other sets, etc.

Set notation:

(3) a. { 5, Lisa, 🐎, 17 }

b. { 5, Lisa, 🐎, { 4, Spiderman, 🐒, 17 }, 🐨 }

Very often, we don't know, or can't specify, the complete membership of a set. Predicate notation is useful in these cases.

(4) {x | x is a natural number}

Read: the set of all x such that x is a natural number

Summary on sets

Set membership:
To indicate that an object is a member of a set, we use a rounded lowercase Greek $\in$. (We will mark the empty set as $\varnothing$. $\varnothing$ is a subset of every set.)

(5) a. Noam Chomsky $\in$ {y | y is a linguist}
Noam Chomsky is a member of the set of all y such that y is a linguist.

b. Stephen Hawking $\notin$ {y | y is a linguist}
Stephen Hawking is not a member of the set of all y such that y is a linguist.

Sets are unordered:
The members of a set are not ordered in any way.

(6) {Chomsky, Hawking} = {Hawking, Chomsky}

No repetitions:
When specifying a set, repetitions of the same object are meaningless.

(7) {Chomsky} = {Chomsky, Chomsky} = {Chomsky, Chomsky, Chomsky}

Summary on sets

Set operations:

1. Some set intersection: The intersection of a set A with a set B is the set of all things that are in both A and B. In symbols, $A \cap B$.

   (8) $A \cap B \coloneqq \{ x \mid x \in A \text{ and } x \in B\}$

2. Set union: The union of a set A with a set B is the set of all things that are in A or B. (Things in both sets are included in the union.)

   (9) $A \cup B \coloneqq \{ x \mid x \in A \text{ or } x \in B\}$

3. Set-theoretic difference: The difference between two sets A and B is the set of all elements in A which are not in B. When the first argument to $-$ is the universe U, this is called complementation.

   (10) $A - B \coloneqq \{ x \mid x \in A \text{ and } x \notin B\}$

4. Subset: The subset relation doesn't return a new set. Rather, it returns truth or falsity.

   (11) $A \subseteq B$ iff for all x, if $x \in A$, then $x \in B$

Summary on relations and functions

A relation is a set of n-tuples $\langle\rangle$. An ordered n-tuple is a finite sequence of n objects of any kind. (12) e.g. could be representation of *admire*.

(12) $\{\langle$Chomsky, Hawking$\rangle$, $\langle$Spiderman, Hulk$\rangle$, $\langle$Homer, Lisa$\rangle$,$\}$

Ordered tuples are ordered, repetitions are meaningful:

(13) a. $\langle$Chomsky, Hawking$\rangle \neq \langle$Hawking, Chomsky$\rangle$

b. $\langle$Chomsky$\rangle \neq \langle$Chomsky, Chomsky$\rangle$

Predicate notation:

(14) $\{\langle x,y\rangle \mid x \text{ admires } y\}$

A relation between two sets is a subset of the *Cartesian product* of the sets:

(15) $A \times B := \{\langle x,y\rangle \mid x \in A \text{ and } y \in B\}$

(16) $\{a,b\} \times \{1,2\} = \left\{ \begin{array}{l} \langle a,1\rangle,\langle a,2\rangle, \\ \langle b,1\rangle,\langle b,2\rangle \end{array} \right\}$

Summary on relations and functions

Relations are not restricted to whether an element in one set is related to one, many, or no element in the other set. Functions, in contrast, map an element in one set (domain) onto at most one element in the other set (range).

(17) relation, function:

$$\begin{tikzcd}
a \arrow[r] & 1 \\
b \arrow[r] & 2 \\
c \arrow[dr] & 3 \\
d \arrow[r] & 4 \\
 & 5
\end{tikzcd}$$

(18) relation, ~~function~~:

$$\begin{tikzcd}
a \arrow[r] & 1 \\
b \arrow[r] & 2 \\
c \arrow[dr] \arrow[ddr] & 3 \\
d & 4 \\
 & 5
\end{tikzcd}$$

(19) a. A relation R is a *function* iff each x in the domain of R is mapped by R to at most one element in the range of R.

b. A function f is *total* iff every element in the domain of f has a value in the range of f. If f fails to meet this condition, it is called a *partial* function.

We gloss $f : \mathrm{A} \mapsto \mathrm{B}$ as ‘the function f with domain A and range B’.

Summary on relations and functions

List notation for functions:

(20) $F := \{\langle Lisa, 1\rangle, \langle Bart, 1\rangle, \langle Mr.Burns, 0\rangle\}$

Table notation for functions:

(21) $\begin{bmatrix} \text{Lisa} & \rightarrow 1 \\ \text{Bart} & \rightarrow 1 \\ \text{Mr. Burns} & \rightarrow 0 \end{bmatrix}$

There is a close correspondence between functions into the domain of truth values {T, F} and sets: for each function into {T, F}, we can form its characteristic set. For each set, we can form its characteristic function.

(22) a. The **characteristic function** of a set A is a function f such that, for any $x \in A, f(x) = 1$ and for any $x \notin A, f(x) = 0$.

b. Let f be a function with range {0,1}. Then, the **set is characterized** by $f = \{x \in D : f(x) = 1\}$

Summary on relations and functions

Let us come back to our original example:

(23) $[\![\text{sleep}]\!]$:=

a. function-talk: *sleep(x)* = 1 iff *x sleeps*

b. set-talk: {x | x is a sleeper}

In set-notation, the concept of set-membership is central to truth.

In function-notation, predicate saturation – involving the application of an argument to the characteristic function – yields truth.

The λ-calculus directly uses the second method (via function application) as the semantic glue for combining sentence constituents.

But it is useful to remember that function application essentially involves the concept of set membership, with respect to the set that the function characterizes.

Summary on Predicate Logic

A formal system is a syntactic object, a set of expressions and rules of combination and derivation. We use Predicate Logic as a formal system to analyze natural languages.

The vocabulary of Predicate Logic:

1. individual constants: $\{d, n, j, ...\} \rightarrow$ *terms*
2. individual variables: $\{x, y, z, ...\}$ $\rightarrow$ *terms*
3. predicate constants: $\{C, D, L, ...\} \rightarrow$ Each predicate has a fixed number of arguments, called its *arity* or *valence*. As we will see, this corresponds closely to argument positions for natural language predicates.
4. connectives: $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$

Connective	Syntax	English
$\neg$	$\neg p$	it is not the case that p
$\wedge$	$p \wedge q$	p and q
$\vee$	$p \vee q$	p and/or q
$\vee_e$	$p \vee_e q$	p or q but not both
$\rightarrow$	$p \rightarrow q$	if p, then q
$\leftrightarrow$	$p \leftrightarrow q$	p if and only if q

5. auxiliary symbols: ()[],
6. formulas: well-formed expressions of the logical language

Summary on Predicate Logic

A toy language L_x:
Let us consider a very simple Predicate Logic language with basic expressions of three categories: names, one-place predicates, and two-place predicates.

(24) Basic expressions of L_x

Category	Basic expressions	NL counterpart
Names	d, n, j, m	Dee, Nat, Jean, Mo
one place predicates	H, C	happy, cries
two place predicates	D, L	dislike, love

(25) Syntactic rules of L_x

a. If δ is a one-place predicate and α is a name, then $\delta(\alpha)$ is a formula.

b. If γ is a two-place predicate and α and β are names, then $\gamma(\alpha, \beta)$ is a formula.

c. If ϕ is a formula, then $\neg\phi$ is a formula.

d. If ϕ and ψ are formulas, then $[\phi \wedge \psi]$, $[\phi \vee \psi]$, $[\phi \rightarrow \psi]$, and $[\phi \leftrightarrow \psi]$ are formulas.

e. Nothing else is a formula.

Summary on Predicate Logic

(26) Syntactic rules of L_x

a. If δ is a one-place predicate and α is a name, then $\delta(\alpha)$ is a formula.

b. If γ is a two-place predicate and α and β are names, then $\gamma(\alpha, \beta)$ is a formula.

c. If ϕ is a formula, then $\neg\phi$ is a formula.

d. If ϕ and ψ are formulas, then $[\phi \wedge \psi]$, $[\phi \vee \psi]$, $[\phi \rightarrow \psi]$, and $[\phi \leftrightarrow \psi]$ are formulas.

e. Nothing else is a formula.

(27) Determine the well-formedness of the following formulas:

a.	$C(j)$	**well-formed**
b.	$D(n, d)$	**well-formed**
c.	$C(j \wedge d)$	**not well-formed**
d.	$L(n, j) \vee C(j)$	**well-formed**
e.	$[C(d) \wedge L(n, j)] \rightarrow H(n)$	**well-formed**

Summary on Predicate Logic

Now we have to talk about the relation between the formal system (predicate logic) and the models that can be used to interpret it, i.e., to assign extra-linguistic entities as the meanings of expressions.

A model *M* is a pair $\langle D, I \rangle$, where *D* is the domain, a set of individuals, and *I* is an interpretation function: as assignment of semantic values to every basic expression (constant) in the language.

Models are distinguished both by the objects in their domains and by the values assigned to the expressions of the language by *I* – by the particular way that the words of the language are "linked" to the things in the world. For example:

(28) $M = \langle D, I \rangle$, where:

a. $D = \{Dee, Nat, Jean, Mo\}$

b. *I* determines the following mapping between names and predicate terms in L_x and objects in *D*:

name	value	predicate	value
d	Dee	H	$\{Nat, Mo\}$
n	Nat	C	$\{Nat, Mo, Dee\}$
j	Jean	D	$\{\langle Mo, Dee \rangle, \langle Nat, Dee \rangle\}$
m	Mo	L	$\{\langle Nat, Jean \rangle, \langle Dee, Jean \rangle, \langle Mo, Jean \rangle\}$

The interpretation of an arbitrary expression α relative to *M*, $[\![\alpha]\!]^M$, is built up recursively on the basis of the basic interpretation function *I* and a set of composition rules: the"engine" of the semantics.

Summary on Predicate Logic

The interpretation of a formula (sentence) of the language is a truth value, where truth values of particular sentences are ultimately determined by the model.

(29) Semantic rules of L_x[1]

a. If δ is a one-place predicate and α is a name, then $[\![\delta(\alpha)]\!]^M = 1$ iff $[\![\alpha]\!]^M \in [\![\delta]\!]^M$.

b. If γ is a two-place predicate and α and β are names, then $[\![\gamma(\alpha,\beta)]\!]^M = 1$ iff $\langle [\![\alpha]\!]^M, [\![\beta]\!]^M \rangle \in [\![\gamma]\!]^M$.

c. If ϕ is a formula, then $[\![\neg\phi]\!]^M = 1$ iff $[\![\phi]\!]^M = 0$.

d. If ϕ and ψ are formulas, then $[\![\phi \wedge \psi]\!]^M = 1$ iff both $[\![\phi]\!]^M = 1$ and $[\![\psi]\!]^M = 1$.

e. If ϕ and ψ are formulas, then $[\![\phi \vee \psi]\!]^M = 1$ iff at least one of $[\![\phi]\!]^M$, $[\![\psi]\!]^M = 1$.

f. If ϕ and ψ are formulas, then $[\![\phi \rightarrow \psi]\!]^M = 1$ iff either $[\![\phi]\!]^M = 0$ or $[\![\psi]\!]^M = 1$.

g. If ϕ and ψ are formulas, then $[\![\phi \leftrightarrow \psi]\!]^M = 1$ iff $[\![\phi]\!]^M = [\![\psi]\!]^M$.

h. Nothing else is a formula.

[1] Reminder: Truth tables for the connectives

p	q	$(p \wedge q)$
1	1	1
1	0	0
0	1	0
0	0	0

p	q	$(p \vee q)$
1	1	1
1	0	1
0	1	1
0	0	0

p	q	$p \rightarrow q$
1	1	1
1	0	0
0	1	1
0	0	1

p	q	$(p \leftrightarrow q)$
1	1	1
1	0	0
0	1	0
0	0	1

Summary on Predicate Logic

(30) Semantic rules of L$_x$

a. If δ is a one-place predicate and α is a name, then $[\![\delta(\alpha)]\!]^M = 1$ iff $[\![\alpha]\!]^M \in [\![\delta]\!]^M$.

b. If γ is a two-place predicate and α and β are names, then $[\![\gamma(\alpha,\beta)]\!]^M = 1$ iff $\langle [\![\alpha]\!]^M, [\![\beta]\!]^M \rangle \in [\![\gamma]\!]^M$.

c. If ϕ is a formula, then $[\![\neg\phi]\!]^M = 1$ iff $[\![\phi]\!]^M = 0$.

d. If ϕ and ψ are formulas, then $[\![\phi \wedge \psi]\!]^M = 1$ iff both $[\![\phi]\!]^M = 1$ and $[\![\psi]\!]^M = 1$.

e. If ϕ and ψ are formulas, then $[\![\phi \vee \psi]\!]^M = 1$ iff at least one of $[\![\phi]\!]^M$, $[\![\psi]\!]^M = 1$.

f. If ϕ and ψ are formulas, then $[\![\phi \rightarrow \psi]\!]^M = 1$ iff either $[\![\phi]\!]^M = 0$ or $[\![\psi]\!]^M = 1$.

g. If ϕ and ψ are formulas, then $[\![\phi \leftrightarrow \psi]\!]^M = 1$ iff $[\![\phi]\!]^M = [\![\psi]\!]^M$.

h. Nothing else is a formula.

Determine the semantic values of the (well-formed) formulas!

(31) a. $C(j)$ **false**

b. $D(n,d)$ **true**

c. $C(j \wedge d)$ **not well-formed**

d. $L(n,j) \vee C(j)$ **true**

e. $[C(d) \wedge L(n,j)] \rightarrow H(n)$ **true**

name	value	pred	value
d	Dee	H	$\{Nat, Mo\}$
n	Nat	C	$\{Nat, Mo, Dee\}$
j	Jean	D	$\{\langle Mo, Dee\rangle, \langle Nat, Dee\rangle\}$
m	Mo	L	$\{\langle Nat, Jean\rangle, \langle Dee, Jean\rangle, \langle Mo, Jean\rangle\}$

From Predicate Logic to Natural Languages

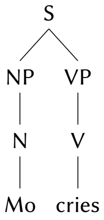
In predicate logic, the semantic rules for determining truth values for formulas were stated in terms of sets. We can do the same thing for natural language sentences.

(32) a. Let $[\![\textit{cries}]\!]^M$ be the set of individuals who cry according to model *M*.

b. Then $[\![\text{Mo cries}]\!]^M = 1$ iff $[\![\textit{Mo}]\!]^M \in [\![\textit{cries}]\!]^M$

Given that the syntax of English is different from the syntax of L_x , we need to relativize our interpretation rules to reflect the structures that serve as the input to our semantic calculations. We call them **composition rules**.

(33)



(34) Given the tree in (33):

a. **Terminal node rule:** the denotation for a terminal node comes from the lexicon

b. **Non-branching node rule:** If α is a node whose only daughter is β, then $[\![\alpha]\!]^M = [\![\beta]\!]^M$

c. **Binary-branching node rule:** If α is a node whose daughter nodes are β and γ, where β is a name and γ is a one-place predicate, then $[\![\alpha]\!]^M = 1$ iff $[\![\beta]\!]^M \in [\![\gamma]\!]^M$

From Predicate Logic to Natural Languages

(35) a. **Terminal node rule:** the denotation for a terminal node comes from the lexicon

b. **Non-branching node rule:** If α is a node whose only daughter is β, then $[\![\alpha]\!]^M = [\![\beta]\!]^M$

c. **Binary-branching node rule:** If α is a node whose daughter nodes are β and γ, where β is a name and γ is a one-place predicate, then $[\![\alpha]\!]^M = 1$ iff $[\![\beta]\!]^M \in [\![\gamma]\!]^M$

Problem: The third rule is specific to one-place predicates. We could try to write a new rule to take care of two-place predicates, but this already becomes somewhat cumbersome.

This (in part) motivates Heim & Kratzer to instead make use of functions rather than sets to describe the meanings of predicates.

(36) a. Let a one-place predicate denote a function from elements in the domain of individuals in our model (D_M) to truth values.

b. $[\![\text{cries}]\!]^M = f : D_M \rightarrow \{1, 0\}$ such that for all $x \in D_M . f(x) = 1$ iff x cries

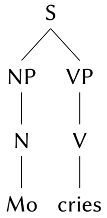
Functional application

(37) a. Let a one-place predicate denote a function from elements in the domain of individuals in our model (D_M) to truth values.

b. $[\![\text{cries}]\!]^M = f : D_M \rightarrow \{1, 0\}$ such that for all $x \in D_M . f(x) = 1$ iff x cries

We redefine **composition rules** in function notation (only the third rule is different):

(38)



(39) Given the tree in (38):

a. **Terminal node rule:** the denotation for a terminal node comes from the lexicon

b. **Non-branching node rule:** If α is a node whose only daughter is β, then $[\![\alpha]\!]^M = [\![\beta]\!]^M$

c. **Function application:** If α is a node whose daughter nodes are β and γ, where $[\![\gamma]\!]^M$ is a function whose domain contains $[\![\beta]\!]^M$, then $[\![\alpha]\!]^M = [\![\gamma]\!]^M([\![\beta]\!]^M)$

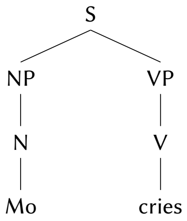
Functional application

(40) a. **Terminal node rule:** the denotation for a terminal node comes from the lexicon

b. **Non-branching node rule:** If α is a node whose only daughter is β, then $[\![\alpha]\!]^M = [\![\beta]\!]^M$

c. **Function application:** If α is a node whose daughter nodes are β and γ, where $[\![\gamma]\!]^M$ is a function whose domain contains $[\![\beta]\!]^M$, then $[\![\alpha]\!]^M = [\![\gamma]\!]^M([\![\beta]\!]^M)$

(41) Let $\alpha =$



Functional application

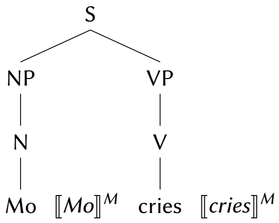
(42) a. **Terminal node rule:** the denotation for a terminal node comes from the lexicon

b. **Non-branching node rule:** If α is a node whose only daughter is β, then $[\![\alpha]\!]^M = [\![\beta]\!]^M$

c. **Function application:** If α is a node whose daughter nodes are β and γ, where $[\![\gamma]\!]^M$ is a function whose domain contains $[\![\beta]\!]^M$, then $[\![\alpha]\!]^M = [\![\gamma]\!]^M([\![\beta]\!]^M)$

1 terminal node rule

(43) Let $\alpha =$



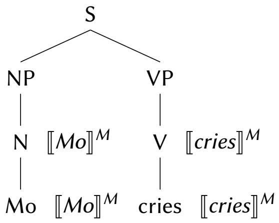
Functional application

(44) a. **Terminal node rule:** the denotation for a terminal node comes from the lexicon

b. **Non-branching node rule:** If α is a node whose only daughter is β, then $[\![\alpha]\!]^M = [\![\beta]\!]^M$

c. **Function application:** If α is a node whose daughter nodes are β and γ, where $[\![\gamma]\!]^M$ is a function whose domain contains $[\![\beta]\!]^M$, then $[\![\alpha]\!]^M = [\![\gamma]\!]^M([\![\beta]\!]^M)$

(45) Let $\alpha =$



1. terminal node rule
2. non-branching node rule

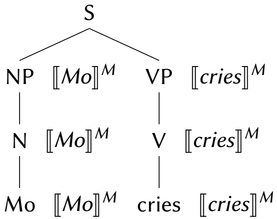
Functional application

(46) a. **Terminal node rule:** the denotation for a terminal node comes from the lexicon

b. **Non-branching node rule:** If α is a node whose only daughter is β, then $[\![\alpha]\!]^M = [\![\beta]\!]^M$

c. **Function application:** If α is a node whose daughter nodes are β and γ, where $[\![\gamma]\!]^M$ is a function whose domain contains $[\![\beta]\!]^M$, then $[\![\alpha]\!]^M = [\![\gamma]\!]^M([\![\beta]\!]^M)$

(47) Let $\alpha =$



1. terminal node rule
2. non-branching node rule
3. non-branching node rule

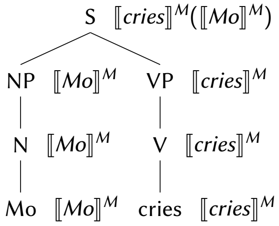
Functional application

(48) a. **Terminal node rule:** the denotation for a terminal node comes from the lexicon

b. **Non-branching node rule:** If α is a node whose only daughter is β, then $[\![\alpha]\!]^M = [\![\beta]\!]^M$

c. **Function application:** If α is a node whose daughter nodes are β and γ, where $[\![\gamma]\!]^M$ is a function whose domain contains $[\![\beta]\!]^M$, then $[\![\alpha]\!]^M = [\![\gamma]\!]^M([\![\beta]\!]^M)$

(49) Let $\alpha =$



1. terminal node rule
2. non-branching node rule
3. non-branching node rule
4. function application

Functional application

By the terminal and non-branching node rule and function application, we get:

(50) $[\![\alpha]\!]^M = [\![\textit{cries}]\!]^M([\![\textit{Mo}]\!]^M)$

Now by substituting the lexical items for their denotations, we get:

(51) $[\![\alpha]\!]^M = [f : D_M \rightarrow \{1, 0\}$ such that for all $x \in D_M, f(x) = 1$ iff x cries $](\text{Mo})$

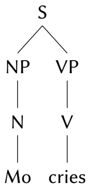
Thus, $[\![\alpha]\!]^M$ denotes the value of the function denoted by $[\![\textit{cries}]\!]^M$ for the argument denoted by $[\![\textit{Mo}]\!]^M$ (i.e. a truth value). Now, recall I from $M = \langle D, I\rangle$ (our model):

name	value	predicate	value
d	Dee	H	$\{Nat, Mo\}$
n	Nat	C	$\{Nat, Mo, Dee\}$
j	Jean	D	$\{\langle Mo, Dee\rangle, \langle Nat, Dee\rangle\}$
m	Mo	L	$\{\langle Nat, Jean\rangle, \langle Dee, Jean\rangle, \langle Mo, Jean\rangle\}$

According to I, the function $[\![\textit{cries}]\!]^M$ returns 1 if applied to $[\![\textit{Mo}]\!]^M$:

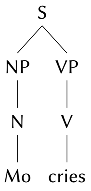
(52) $$[\![\alpha]\!]^M = \begin{bmatrix} \text{Nat} & \rightarrow & 1 \\ \text{Mo} & \rightarrow & 1 \\ \text{Dee} & \rightarrow & 1 \\ \text{Jean} & \rightarrow & 0 \end{bmatrix}(Mo) = 1$$

Functional application

(53)


$$\left[\!\left[\text{S} \right]\!\right]^{M} = \begin{bmatrix} \text{Nat} & \rightarrow & 1 \\ \text{Mo} & \rightarrow & 1 \\ \text{Dee} & \rightarrow & 1 \\ \text{Jean} & \rightarrow & 0 \end{bmatrix} (Mo) = 1$$

In (53), we defined the function by displaying it in a table with the information coming from *M*, thus we end up with a mere truth value. Often we don't have enough world knowledge (or simply space) to give the table notation, i.e. we don't know of every individual whether (s)he cries. Hence, we end up with truth conditions, rather than a truth value in (54). (53) is equivalent to (54).

(54)


$$\left[\!\left[\text{S} \right]\!\right]^{M} = \begin{bmatrix} f : D_M \rightarrow \{1, 0\} \text{ such that} \\ \text{for all } x \in D_M . f(x) = 1 \text{ iff x cries} \end{bmatrix} (Mo) = 1 \text{ iff Mo cries}$$

Schönfinkelization

There is an incompatibility between the kind of step-wise, local function application we have just done and the way in which we have looked at predicate-valency (one-place, two-place, three-place, etc.) earlier.

We have treated intransitive verbs as one-place predicates denoting a set of individuals. We have treated transitive verbs as two-place predicates denoting a set of ordered pairs. Consider a version of our current model M' in which Dee, Nat, and Mo are the only individuals:

(55) $[\![\textit{dislike}]\!]^{M'} = \{\langle Nat, Dee\rangle, \langle Mo, Dee\rangle\}$

(56) $$[\![\textit{dislike}]\!]^{M'} = \begin{bmatrix} \langle \text{Nat, Dee}\rangle & \rightarrow & 1 \\ \langle \text{Nat, Mo}\rangle & \rightarrow & 0 \\ \langle \text{Nat, Nat}\rangle & \rightarrow & 0 \\ \langle \text{Dee, Nat}\rangle & \rightarrow & 0 \\ \langle \text{Dee, Mo}\rangle & \rightarrow & 0 \\ \langle \text{Dee, Dee}\rangle & \rightarrow & 0 \\ \langle \text{Mo, Mo}\rangle & \rightarrow & 0 \\ \langle \text{Mo, Dee}\rangle & \rightarrow & 1 \\ \langle \text{Mo, Nat}\rangle & \rightarrow & 0 \end{bmatrix}$$

Schönfinkelization

(57) $$[\![\textit{dislike}]\!]^{M'} = \begin{bmatrix} \langle \text{Nat, Dee} \rangle & \rightarrow & 1 \\ \langle \text{Nat, Mo} \rangle & \rightarrow & 0 \\ \langle \text{Nat, Nat} \rangle & \rightarrow & 0 \\ \langle \text{Dee, Nat} \rangle & \rightarrow & 0 \\ \langle \text{Dee, Mo} \rangle & \rightarrow & 0 \\ \langle \text{Dee, Dee} \rangle & \rightarrow & 0 \\ \langle \text{Mo, Mo} \rangle & \rightarrow & 0 \\ \langle \text{Mo, Dee} \rangle & \rightarrow & 1 \\ \langle \text{Mo, Nat} \rangle & \rightarrow & 0 \end{bmatrix}$$

(58) $[\![\textit{dislike}]\!]^{M'} = \{\langle \textit{Nat}, \textit{Dee} \rangle, \langle \textit{Mo}, \textit{Dee} \rangle\}$

Problems:

1. binary branching: in the syntax, transitive verbs combine with the direct object to form a VP, and VPs combine with the subject to form a sentence
2. locality: semantic interpretation rules are local, the denoation of any non-terminal node is computed from the denotation of its daughter nodes
3. Frege's conjecture: semantic composition is functional application

What we need, then, is a way to represent all n-place predicates as one-place predicates. This is exactly what **Schönfinkelization** allows us to do.

Schönfinkelization

Schönfinkelization does the following: It takes an n-place function and converts it into a nested/complex one-place function. In other words: $f(\langle a, b\rangle) = (f'(a))(b)$, but also: $f(\langle a, b\rangle) = (f''(b))(a)$

This one-place function combines with one of the arguments (in our ordered pair) to yield another one-place function, which combines with another one of the arguments in the pair.... to yield a truth-value (the order of arguments is not important).

(59) $[\![\textit{dislike}]\!]^{M'} = \{\langle \textit{Nat}, \textit{Dee}\rangle, \langle \textit{Mo}, \textit{Dee}\rangle\}$

(60) $$[\![\text{dislike}]\!]^{M'} = \begin{bmatrix} \text{Nat} \rightarrow \begin{bmatrix} \text{Dee} \rightarrow 1 \\ \text{Mo} \rightarrow 0 \\ \text{Nat} \rightarrow 0 \end{bmatrix} \\ \text{Dee} \rightarrow \begin{bmatrix} \text{Nat} \rightarrow 0 \\ \text{Mo} \rightarrow 0 \\ \text{Dee} \rightarrow 0 \end{bmatrix} \\ \text{Mo} \rightarrow \begin{bmatrix} \text{Mo} \rightarrow 0 \\ \text{Dee} \rightarrow 1 \\ \text{Nat} \rightarrow 0 \end{bmatrix} \end{bmatrix}$$

(61) $$[\![\text{dislike}]\!]^{M'} = \begin{bmatrix} \text{Nat} \rightarrow \begin{bmatrix} \text{Dee} \rightarrow 0 \\ \text{Mo} \rightarrow 0 \\ \text{Nat} \rightarrow 0 \end{bmatrix} \\ \text{Dee} \rightarrow \begin{bmatrix} \text{Nat} \rightarrow 1 \\ \text{Mo} \rightarrow 1 \\ \text{Dee} \rightarrow 0 \end{bmatrix} \\ \text{Mo} \rightarrow \begin{bmatrix} \text{Mo} \rightarrow 0 \\ \text{Dee} \rightarrow 0 \\ \text{Nat} \rightarrow 0 \end{bmatrix} \end{bmatrix}$$

We can *schönfinkel* from **left-to-right** (first argument in outer bracket) or from **right-to-left** (second argument in outer bracket). Which one do you think is which?

Schönfinkelization

(62) $[\![\textit{dislike}]\!]^{M'} = \{\langle Nat, Dee\rangle, \langle Mo, Dee\rangle\}$

Left-to-right:

(63) $$[\![\text{dislike}]\!]^{M'} = \begin{bmatrix} \text{Nat} \rightarrow \begin{bmatrix} \text{Dee} \rightarrow 1 \\ \text{Mo} \rightarrow 0 \\ \text{Nat} \rightarrow 0 \end{bmatrix} \\ \text{Dee} \rightarrow \begin{bmatrix} \text{Nat} \rightarrow 0 \\ \text{Mo} \rightarrow 0 \\ \text{Dee} \rightarrow 0 \end{bmatrix} \\ \text{Mo} \rightarrow \begin{bmatrix} \text{Mo} \rightarrow 0 \\ \text{Dee} \rightarrow 1 \\ \text{Nat} \rightarrow 0 \end{bmatrix} \end{bmatrix}$$

Right-to-left:

(64) $$[\![\text{dislike}]\!]^{M'} = \begin{bmatrix} \text{Nat} \rightarrow \begin{bmatrix} \text{Dee} \rightarrow 0 \\ \text{Mo} \rightarrow 0 \\ \text{Nat} \rightarrow 0 \end{bmatrix} \\ \text{Dee} \rightarrow \begin{bmatrix} \text{Nat} \rightarrow 1 \\ \text{Mo} \rightarrow 1 \\ \text{Dee} \rightarrow 0 \end{bmatrix} \\ \text{Mo} \rightarrow \begin{bmatrix} \text{Mo} \rightarrow 0 \\ \text{Dee} \rightarrow 0 \\ \text{Nat} \rightarrow 0 \end{bmatrix} \end{bmatrix}$$

Which one do you think we use for natural languages?

Schönfinkelization

(65) $[\![\textit{dislike}]\!]^{M'} = \{\langle \textit{Nat}, \textit{Dee}\rangle, \langle \textit{Mo}, \textit{Dee}\rangle\}$

Left-to-right:

(66) $$[\![\text{dislike}]\!]^{M'} = \left[\begin{array}{l} \text{Nat} \rightarrow \left[\begin{array}{l} \text{Dee} \rightarrow 1 \\ \text{Mo} \rightarrow 0 \\ \text{Nat} \rightarrow 0 \end{array}\right] \\ \text{Dee} \rightarrow \left[\begin{array}{l} \text{Nat} \rightarrow 0 \\ \text{Mo} \rightarrow 0 \\ \text{Dee} \rightarrow 0 \end{array}\right] \\ \text{Mo} \rightarrow \left[\begin{array}{l} \text{Mo} \rightarrow 0 \\ \text{Dee} \rightarrow 1 \\ \text{Nat} \rightarrow 0 \end{array}\right] \end{array}\right]$$

Right-to-left:

(67) $$[\![\text{dislike}]\!]^{M'} = \left[\begin{array}{l} \text{Nat} \rightarrow \left[\begin{array}{l} \text{Dee} \rightarrow 0 \\ \text{Mo} \rightarrow 0 \\ \text{Nat} \rightarrow 0 \end{array}\right] \\ \text{Dee} \rightarrow \left[\begin{array}{l} \text{Nat} \rightarrow 1 \\ \text{Mo} \rightarrow 1 \\ \text{Dee} \rightarrow 0 \end{array}\right] \\ \text{Mo} \rightarrow \left[\begin{array}{l} \text{Mo} \rightarrow 0 \\ \text{Dee} \rightarrow 0 \\ \text{Nat} \rightarrow 0 \end{array}\right] \end{array}\right]$$

Which one do you think we use for natural languages? right-to-left

The order of elements in the pairs in (65) is $\langle \textit{subj}, \textit{obj}\rangle$. Since the object combines first with the verb, we feed the right element into the the nested one-place function first. This will give us another one-place predicate to which we feed the subject, i.e. the left element of the pair in (65).

Summary transitive predicates

(68) $[\![\textit{dislike}]\!]^{M'} = \{\langle Nat, Dee\rangle, \langle Mo, Dee\rangle\}$ *set-talk*

(69) $$[\![\text{dislike}]\!]^{M'} = \left[\begin{array}{l} \text{Nat} \rightarrow \left[\begin{array}{l} \text{Dee} \rightarrow 0 \\ \text{Mo} \rightarrow 0 \\ \text{Nat} \rightarrow 0 \end{array}\right] \\ \text{Dee} \rightarrow \left[\begin{array}{l} \text{Nat} \rightarrow 1 \\ \text{Mo} \rightarrow 1 \\ \text{Dee} \rightarrow 0 \end{array}\right] \\ \text{Mo} \rightarrow \left[\begin{array}{l} \text{Mo} \rightarrow 0 \\ \text{Dee} \rightarrow 0 \\ \text{Nat} \rightarrow 0 \end{array}\right] \end{array}\right]$$ *function-talk*

(70) $[\![\text{dislike}]\!]^{M'} = f : D_{M'} \rightarrow \{g : D_{M'} \rightarrow \{1, 0\}\}$ such that for all $x \in D_{M'}, f(x) = g_x : D_{M'} \rightarrow \{1, 0\}$ such that for all $y \in D_{M'}, g_x(y) = 1$ iff y dislikes x *function-talk*

Instead of treating two-place predicates as denoting sets of ordered pairs, we model them as functions from individuals to functions from individuals to truth values (i.e., functions from individuals to one-place predicates).