

0. Announcements

-Read: Chapters 1-3 up to 3.5 (if you haven't already); Chapter 4, up to 4.4.

-**Required attendance:** Efron lecture, April 11th, 4:15PM, Rose Hills Theater: *Finding Order in Chaos: Tonal Variation in East African Languages*.

1. Sets and their characteristic functions (adapted from H&K)

So far, we have treated intransitive predicates as sets of individuals. "This is the standard choice for the extensions of 1-place predicates in logic. The intuition here is that each verb denotes the set of those things that it is true of." (H&K)

Alternatively, predicates can be treated as **functions**.

What are functions? Functions are a special kind of relation. Recall that a 2-place relation is a set of ordered pairs. For functions (as opposed to non-function relations), the second member of each pair is uniquely determined by the first. Here is the definition:

- (1) A relation f is a **function** iff it satisfies the following condition: For any x : If there are y and z such that $\langle x, y \rangle \in f$ and $\langle x, z \rangle \in f$, then $y = z$.

Functions, like sets, can be defined in various ways. We can simply **list** the function's elements, as in (2), or we can use a table, as in (3).

(2) $F := \{ \langle \text{Tasha}, 1 \rangle, \langle \text{Nick}, 0 \rangle \}$

(3)

F :=	Tasha	→	1
	Nick	→	0

(We will use the **λ-notation** defined below.)

Each function has a **domain** and a **range**, which are the sets defined as follows:

- (4) Let f be a function. Then the **domain** of f is $\{x \mid \text{there is a } y \text{ such that } f(x)=y\}$, and the **range** of f is $\{y \mid \text{there is a } x \text{ such that } f(x)=y\}$.

E.g., the domain of the above function is $\{\text{Tasha}, \text{Nick}\}$, and the range is $\{0, 1\}$.

When A is the domain and B the range of f , we also say that f is **from** A and **onto** B . If C is a superset of f 's range, we say that f is **into** (or **to**) C .

For " f is from A (in)to B ", we write " $f \mid A \rightarrow B$ ".

The uniqueness condition built into the definition of function-hood ensures that whenever f is a function and x a member of its domain, the following definition makes sense:

(5) $f(x) :=$ the unique y such that $\langle x, y \rangle \in f$.

For " $f(x)$ ", read " f of x ". $f(x)$ is also called the "value of f for the argument x ", and we say that f maps x to y ".

Functions with large or infinite domains are often defined by specifying a **condition** that is to be met by each argument-value pair. For example:

(6) $F_{\text{mother-of}} := f: D \rightarrow D$
For every $x \in D, f(x) = x$'s mother
(D is the set of all actual individuals.)

Read as "F_{mother-of} is to be that function f from D into D such that, for every x in D , $f(x) = x$'s mother."

There exists a one-to-one correspondence between sets and certain functions:

- (7) Let A be a set. Then the **characteristic function** of A is that function f such that, for any $x \in A, f(x) = 1$, and for any $x \notin A, f(x) = 0$.

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

Intransitive predicates, then, can be taken to denote **functions**:

(9) $\llbracket \text{female} \rrbracket^g = f: D \rightarrow \{0, 1\}$
For all $x \in D, f(x) = 1$ iff female(x)

(10) $\llbracket \text{run} \rrbracket^g = f: D \rightarrow \{0, 1\}$
For all $x \in D, f(x) = 1$ iff run(x)

From here on out, we will define our functions using the **λ -notation**:

$$(11) \llbracket female \rrbracket^g = [\lambda x : x \in D . female(x)]$$

The λ -term here reads as “the function that maps every x in D to 1 iff $female(x)$.”

For the above λ -terms, we say that “ x ” is the **argument variable**, “ $x \in D$ ” is the **domain condition**. The domain condition is introduced by a colon, and the value description by a period.

The domain condition delimits the function’s domain by restricting possible values for the argument variable. For example, above, it states that the argument must be a member of D , the set of individuals.

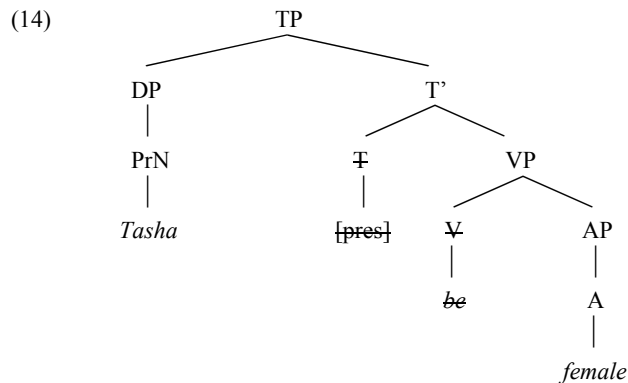
Since we use “ x ” to stand for arbitrary individuals in D , we will often leave out the domain condition:

$$(12) \llbracket female \rrbracket^g = [\lambda x . female(x)]$$

This function can apply to an individual, yielding a truth-value:

$$(13) \llbracket female \rrbracket^g = [\lambda x . female(x)](Tasha) = 1 \text{ iff } female(Tasha)$$

Now, returning to our syntactic trees:



Given our treatment of *female* as a function, we can posit the following rules:

For any assignment g :

- (a) *Functional Application* (FA)
If α is a branching node, $\{\beta, \gamma\}$ are its daughters, and $\llbracket \beta \rrbracket^g$ is a function whose domain contains $\llbracket \gamma \rrbracket^g$, then $\llbracket \alpha \rrbracket^g = \llbracket \beta \rrbracket^g (\llbracket \gamma \rrbracket^g)$.
- (b) *Non-Branching Nodes* (NN)
If α is a non-branching node whose daughter node is β , then $\llbracket \alpha \rrbracket^g = \llbracket \beta \rrbracket^g$.
- (c) *Terminal Nodes* (TN)
If α is a terminal node, then $\llbracket \alpha \rrbracket^g$ is specified in the lexicon.

Example derivation:

$\llbracket TP \rrbracket^g$	
$\llbracket T' \rrbracket^g (\llbracket DP \rrbracket^g)$	(a)
$\llbracket T' \rrbracket^g (\llbracket Tasha \rrbracket^g)$	(b)x2
$\llbracket T' \rrbracket^g (Tasha)$	(c)
$\llbracket female \rrbracket^g (Tasha)$	(b)x4
$[\lambda x . female(x)](Tasha)$	(c)
$= 1 \text{ iff } female(Tasha)$	simplification

2. Type-driven interpretation

What do we gain by switching to functions? We will no longer need semantic rules for **specific** types of syntactic constituents (as our former grammar did.)

This approach, which does not make reference to particular types of phrases, is called ‘type-driven’ interpretation. Notice that our **Functional Application** rule does not specify the linear order of β and γ . It’s the semantic types of the daughter nodes that will determine the procedure for calculating the meaning of the mother node.

Our semantic rules, then, only need to see the lexical items and the hierarchical structure in which they are arranged. Syntactic category labels are going to become irrelevant.

3. Revised Class Grammar

(i) Inventory of denotations

Semantic types

- (a) e and t are semantic types.
- (b) If σ and τ are semantic types, then $\langle\sigma, \tau\rangle$ is a semantic type.
- (c) Nothing else is a semantic type.

Semantic denotation domains

- (a) $D_e := D$ (the set of individuals).
- (b) $D_t := \{0, 1\}$ (the set of truth-values).
- (c) For any semantic types σ and τ , $D_{\langle\sigma, \tau\rangle}$ is the set of all functions from D_σ to D_τ .

(ii) Lexicon

For any assignment g :

PrN: $\llbracket Alexis \rrbracket^g = \text{Alexis}, \dots$

Pron: For any index i , $\llbracket she_i \rrbracket^g = g(i)$, $\llbracket he_i \rrbracket^g = g(i)$, $\llbracket him_i \rrbracket^g = g(i)$, $\llbracket her_i \rrbracket^g = g(i)$, ...

Pron_{wh}: *who*.

V: $\llbracket laugh \rrbracket^g = [\lambda x . \text{laugh}(x)]$

A: $\llbracket kind \rrbracket^g = [\lambda x . \text{kind}(x)]$

N: $\llbracket cat \rrbracket^g = [\lambda x . \text{cat}(x)]$

P: $\llbracket around \rrbracket^g = [\lambda x . \text{around}(x)]$

V_t: $\llbracket love \rrbracket^g = [\lambda x . [\lambda y . \text{love}(x)(y)]]$

A_t: $\llbracket fond \rrbracket^g = [\lambda x . [\lambda y . \text{fond}(x)(y)]]$

N_t: $\llbracket part \rrbracket^g = [\lambda x . [\lambda y . \text{part}(x)(y)]]$

P_t: $\llbracket above \rrbracket^g = [\lambda x . [\lambda y . \text{above}(x)(y)]]$

D_q: $\llbracket every \rrbracket^g =$

$\llbracket no \rrbracket^g =$

$\llbracket a \rrbracket^g =$

$\llbracket some \rrbracket^g =$

Neg: $\llbracket not \rrbracket^g =$

Conj: $\llbracket and \rrbracket^g =$

$\llbracket or \rrbracket^g =$

T: (We neglect for now the semantic contribution of tense.)

Semantically vacuous: V_{be}: *be*; D_a: *a*; P_{of}: *of*; C: *that*.

(iii) Syntax

TP \rightarrow DP T'

T' \rightarrow T VP

DP \rightarrow PrN/Pron

DP \rightarrow D NP

VP \rightarrow V (DP)

PP \rightarrow P (DP)

AP \rightarrow A (PP)

NP \rightarrow N (PP)

VP \rightarrow V_{be} AP/PP/DP

NP \rightarrow AP NP

NP \rightarrow NP PP

XP \rightarrow Neg XP

Where $X \in \{V, A, P, D\}$.

(iv) Semantic rules of composition

For any assignment g , and index i ,

(a) Functional Application (FA)

If α is a branching node, $\{\beta, \gamma\}$ are its daughters, and $\llbracket \beta \rrbracket^g$ is a function whose domain contains $\llbracket \gamma \rrbracket^g$, then $\llbracket \alpha \rrbracket^g = \llbracket \beta \rrbracket^g (\llbracket \gamma \rrbracket^g)$.

(b) Non-Branching Nodes (NN)

If α is a non-branching node whose daughter node is β , then $\llbracket \alpha \rrbracket^g = \llbracket \beta \rrbracket^g$.

(c) Terminal Nodes (TN)

If α is a terminal node, then $\llbracket \alpha \rrbracket^g$ is specified in the lexicon.

(d) Predicate Modification (TN)

(e) Predicate Abstraction (PA)

(f) Traces Rule (TR)

$\llbracket t_i \rrbracket^g = g(i)$.

4. Transitives

What kinds of functions can we assign to transitives?

In this case, we want functions that map individuals to functions (from individuals to truth values):

$$(15) \quad \llbracket love \rrbracket^g = [\lambda x . [\lambda y . love(y)(x)]]$$

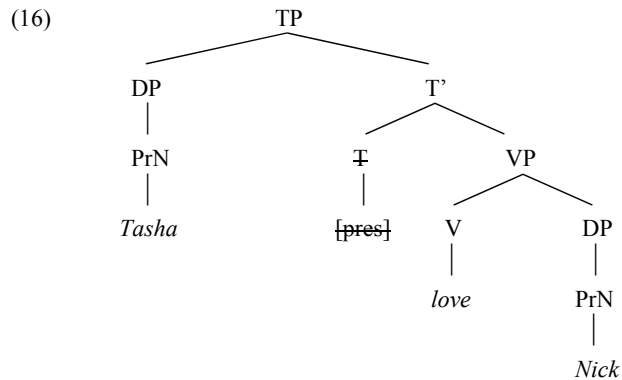
Now we will need two ways to read lambda expressions.

When there is a **statement** to the right of “.”, the function maps the argument to 1 iff the statement is true. (This is the case for intransitives.)

In the case of *love*, the function maps the argument to a **value** (here, a function).

Thus, this is “the function that maps every *x* to the function that maps every *y* to 1 iff *love*(*y*)(*x*).”

(See p. 37).



Exercise 1

Identify the semantic types for each node in (16).

Calculate the truth-conditions for (16).

Exercise 2

Functions can themselves have functions as arguments.

Propose a functional denotation for *not*:

(17) Nick is not female.

Derive the truth-conditions for (17).

Exercise 3

Propose functional denotations for the quantificational determiners.

Derive the following:

(18) Every lion roared.

Exercise 4

Account for the following:

(19) Alexis visited every planet.

Exercise 5

How can/should we account for modifiers?

(20) Kaline is a grey cat.