Lecture 2. Model-theoretic semantics, Lambdas, and NP semantics

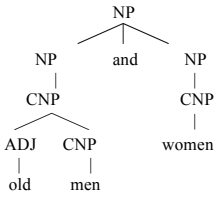
1. Lexical and Structural Ambiguity

Lexical ambiguity: *bank$_1$, bank$_2$* : both CN (common noun), homonyms;
open$_1$ (ADJ), *open$_2$* (IV) (intransitive verb), *open$_3$* (TV) (transitive verb).

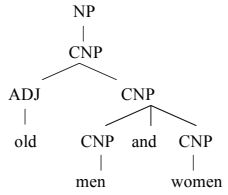
Structural ambiguity. Compositionality requires a "disambiguated language" (a "language without ambiguity"). So we interpret expressions ***with syntactic structure,*** not just strings.

(1) old men and women. Two meanings, two structures. "old" applies only to "men", or to "men and women".

(a)

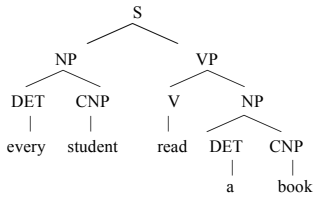


(b)



(2) Every student read a book. (Quantifier scope ambiguity)

Just one (surface) syntactic structure:



Predicate logic representations of the two readings:
(i) $\forall x$ (Student (x) $\rightarrow$ $\exists y$ (Book (y) & Read (x,y))
(ii) $\exists y$ (Book (y) & $\forall x$ (Student (x) $\rightarrow$ Read (x,y))

Compositional interpretation of the English sentence: ??. More below, and more next week..

The difficulty for compositionality if we try to use predicate calculus to represent "logical form": What is the interpretation of "every student"? There is no appropriate syntactic category or semantic type in predicate logic. Inadequacy of 1st-order predicate logic for representing the semantic structure of natural language. We can solve this problem when we have the lambda-calculus and a richer type theory.

Categories of PC:		**Categories of NL:**
Formula	-	Sentence
Predicate	-	Verb, Common Noun, Adjective
Term		
Constant	-	Proper Noun
Variable	-	Pronoun (he, she, it)
==========		
(no more)	-	Verb Phrase, Noun Phrase, Common Noun Phrase, Adjective Phrase, Determiner, Preposition, Prepositional Phrase, Adverb,

2. Lambdas

2.1. A first-order part of the lambda-calculus.

To begin looking at the lambda calculus, we will start with just a "first-order" part of it, as if we were just adding a bit of the lambda calculus to the predicate calculus rules from Lecture 1. Then in section 2.2. we will look at the fully typed lambda calculus as given in Montague's Intensional Logic Rule 7 in the Appendix.

Lambda-abstraction rule, first version.

λ-abstraction applies to ***formulas*** to make ***predicates***. This extends PC in a way that allows us to represent more complex Common Noun Phrases, Adjective Phrases, some Verb Phrases. For some other categories we will need the full version of the λ-abstraction rule.
R9: If $\varphi \in$ Form and v is a variable, then $\lambda v[\varphi] \in$ Pred-1.
S9: $\|\lambda v[\varphi]\|^{M, g}$ is the set S of all $d \in D$ such that $\| \varphi \|^{M, g [d/v]} = 1$.

Examples. For all examples, assume that we start with an assignment g such that g(***v***) = John for all ***v***. In most of the examples below, the choice of initial assignment makes no difference. And assume that I(**b**) = Bill, I(**m**) = Mary.

(i) $\| \boldsymbol{\lambda x}[\mathbf{run}(\boldsymbol{x})] \|^{M, g} =$ the set of all individuals that run.

(ii) $\| \boldsymbol{\lambda x}[\mathbf{love}(\boldsymbol{x}, \mathbf{b})] \|^{M, g} =$ the set of all individuals that love $\| \mathbf{b} \|^{M, g[d/x]}$, i.e. I(**b**), i.e. Bill.

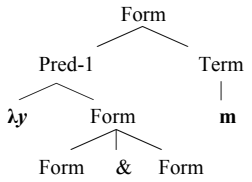
(iii) $\| \boldsymbol{\lambda x}[\mathbf{love}(\boldsymbol{x}, \boldsymbol{y})] \|^{M, g} =$ the set of all individuals that love $\| \boldsymbol{y} \|^{M, g[d/x]}$, i.e. g(***y***), i.e. John.

(iv) $\| \boldsymbol{\lambda x}[\mathbf{fish}(\boldsymbol{x}) \,\&\, \mathbf{love}(\boldsymbol{x}, \mathbf{b})] \|^{M, g} =$ the set of all fish that love Bill.

(v) to represent "walks and talks": $\boldsymbol{\lambda y}[(\mathbf{walk}(\boldsymbol{y}) \,\&\, \mathbf{talk}(\boldsymbol{y}))]$

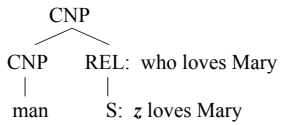
(vi) to represent "Mary walks and talks" with constituents that correspond to surface syntax:
$\boldsymbol{\lambda y}[(\mathbf{walk}(\boldsymbol{y}) \,\&\, \mathbf{talk}(\boldsymbol{y}))]\,(\mathbf{m})$

Syntactic Structure of the formula in (vi):



By λ-conversion (see below), the formula in (vi) is equivalent to (**walk**(**m**) & **talk**(**m**)).

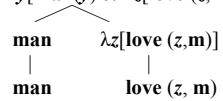
(vii) to represent the CNP "man who loves Mary":
Syntactic structure:



Rule for combining CNP and REL: $\boldsymbol{\lambda y}$[CNP'(***y***) & REL'(***y***)] (combining "translations")

Compositional translation of the syntactic structure above into λ-calculus: (read bottom-to-top)

$\boldsymbol{\lambda y}[\mathbf{man}(\boldsymbol{y}) \,\&\, \boldsymbol{\lambda z}[\mathbf{love}(\boldsymbol{z},\mathbf{m})]\,(\boldsymbol{y})]$



By λ-conversion (see below), the top line is equivalent to: $\boldsymbol{\lambda y}[\mathbf{man}(\boldsymbol{y}) \,\&\, \mathbf{love}(\boldsymbol{y},\mathbf{m})]$

2.2. The typed lambda calculus.

The full version of the typed lambda calculus fits into Montague's intensional logic with its type theory; see the Appendix for a complete statement of Montague's intensional logic. The parts we will use most will be the type theory, the lambda calculus (Rule 7), and the rule of "functional application" (Rule 6). Montague's intensional logic includes the predicate calculus as a subpart (see Rule 2), but not restricted to first-order: we can quantify over variables of any type.

Lambda-abstraction, full version.
In general: λ-expressions denote ***functions.***
λv[α] denotes a function whose ***argument*** is represented by the variable v and whose ***value*** for any given value of v is specified by the expression α.

Example: $\boldsymbol{\lambda x}[\boldsymbol{x}^2 + 1]$ denotes the function $\boldsymbol{x} \rightarrow \boldsymbol{x}^2 + 1$.
Function-argument application: $\boldsymbol{\lambda x}[\boldsymbol{x}^2 + 1]\,(5) = 26$

λ-expressions provide explicit specification of the functions they name, unlike arbitrary names like ***f, g.*** (The λ-calculus was invented by the logician Alonzo Church. The programming language LISP, invented by John McCarthy, was modelled on the λ-calculus.)

Syntactic and Semantic Rule: (a restatement of Syntactic and Semantic Rules 7 of IL)
R7': If α is an expression of any type ***a*** and v is a variable of type ***b***, then λv[α] is an expression of type $\boldsymbol{b} \rightarrow \boldsymbol{a}$ (the type of functions from ***b***-type things to ***a***-type things.)
S7': $\| \lambda v[\alpha] \|^{M, g}$ is that function ***f*** of type $\boldsymbol{b} \rightarrow \boldsymbol{a}$ such that for any object d of type ***b***,
$\boldsymbol{f}(d) = \| \alpha \|^{M, g[d/v]}$.

Lambda-conversion: A principle concerning the application of λ-expressions to arguments.

Examples: $\boldsymbol{\lambda x}[\boldsymbol{x}^2 + 1]\,(5) = 5^2 + 1 = 26$
$\boldsymbol{\lambda x}[\mathbf{run}(\boldsymbol{x})](\mathbf{b}) \equiv \mathbf{run}(\mathbf{b})$
$\boldsymbol{\lambda y}[(\mathbf{walk}(\boldsymbol{y}) \,\&\, \mathbf{talk}(\boldsymbol{y}))]\,(\mathbf{m}) \equiv (\mathbf{walk}(\mathbf{m}) \,\&\, \mathbf{talk}(\mathbf{m}))$
$\boldsymbol{\lambda z}[\mathbf{love}(\boldsymbol{z},\mathbf{m})]\,(\boldsymbol{y}) \equiv \mathbf{love}(\boldsymbol{y},\mathbf{m})$

Lambda-conversion Rule: λv[α](β) ≡ α′, where α′ is like α but with every free occurrence of v replaced by β.
(Note: Occurrences of v that are free in α are bound by λv in λv[α].)

NOTE! You can get **on-line lambda practice** – see Resources page on the course website!

3. Montague's semantics for Noun Phrases.

3.1. Semantics via direct model-theoretic interpretation of English.

[See Larson's chapter 12.]

3.2. Semantics via translation from English into a logical language.

Indicated by translations of English expressions into the λ-calculus. ***P*** is a variable ranging over sets, i.e. a predicate variable. (In full Montague grammar with intensionality, the analysis uses variables over properties, so sometimes in discussion in class, we talk as though ***P*** were a variable over properties.)

John	$\lambda P[P(\mathbf{j})]$
John walks	$\lambda P[P(\mathbf{j})]$ (**walk**) ≡ **walk** (**j**)
every student	$\lambda P[\forall x$ (**student**(x) → $P(x)$)]
every student walks	$\lambda P[\forall x$ (**student**(x) → $P(x)$)] (**walk**) ≡ $\forall x$(**student**(x) → **walk**(x))
a student	$\lambda P[\exists x$ (**student**(x) & $P(x)$)]
the king	$\lambda P[\exists x$ (**king**(x) & $\forall y$ (**king**(y) → $y = x$) & $P(x)$)]
	(the set of properties which the one and only king has)

==========

Appendix: Montague's intensionsal logic, with lambdas and types.

A.1 Introduction

Tools like Montague's Intensional Logic are important in making a more satisfactory compositional analysis of natural language semantics possible. What are the differences between Montague's IL and PC? Here are some of the most important:

(i) The rich type structure of IL.

(ii) The central role played by function-denoting expressions. All of the types except the basic types *e* and *t* are *functional types*, and all of the expressions of IL except those of types *e* and *t* are expressions which denote functions. Functions may serve as the arguments and as the values of other functions. In particular, all relations are also represented as functions.

(iii) The inclusion of the operation of "functional application" or "function-argument application", the application of a function to its argument.

(iv) The use of lambda-expressions. The lambda-operator is the basic tool for building expressions which denote functions.

(v) In place of the one "world" of PC (where there is in effect no distinction between a "world" and a model), the models of IL include a set of *possible worlds*. Possible worlds are crucially connected with intension/extension distinction and with intensional types. Possible worlds, in particular, underlie the interpretation of modal operators and referential opacity.

(vi) The models of IL also include, in one way or another, a structure of time, used among other things in the interpretation of tense operators like **PAST** in the fragment below.

A.2. Intensional Logic (IL).

A.2.1. Types and model structures.

A.2.1.1. Types

Montague's IL is a *typed* intensional language; unlike the predicate calculus, which has variables of only one type (the type of entities or individuals), and expressions only of the types of individuals, truth-values, and n-ary relations over individuals, IL has a rich system of types which makes it much easier to achieve a (relatively) close fit between expressions of various categories of a natural language and expressions of IL. The types serve as syntactic categories for the expressions of IL; because of the role of IL as an intermediate language in the semantic interpretation of natural language, the same types are referred to as *semantic types* for expressions of natural language.

The types of Montague's IL are as follows:

Basic types: e (entities), t (truth values)

Functional types: If a,b are types, then <a,b> is a type (the type of functions from a-type things to b-type things.) **Note:** We use interchangeably **the two notations <a,b> and a → b**, both of which are common in the literature.

Intensional types: If a is a type, then <s,a> is a type (the type of functions from possible worlds to things (extensions) of type a.)

(In some systems, the basic type t is taken as intensional, interpreted as the type of propositions rather than of truth-values. In general, we will mostly ignore intensionality in these lectures, working most of the time with extensional versions of our fragments and mentioning intensionality only where directly relevant. But that is only for simplicity of exposition; in general, a thoroughly intensional semantics is presupposed.)

A.2.1.2. Model structures.

In the first lecture, we introduced the simple model structure **M1** for interpreting the predicate calculus. A model **M** for the typed intensional logic IL has much more structure, but that structure is built up recursively from a small set of primitives.

Model structure for IL: **M** = <D, W, ≤, I>. Each model must contain:

A domain D of entities (individuals)

A set W of possible worlds (or possible world-time pairs, or possible situations)

≤ : an ordering (understood as temporal order) on W

I: Interpretation function which assigns semantic values to all constants.

The domains of possible denotations for expressions of type *a* (relative to D,W) are defined recursively as follows:

$\mathbf{D}_e$ = D

$\mathbf{D}_t$ = {0,1}

$\mathbf{D}_{<a,b>}$ = {f | f: $\mathbf{D}_a \rightarrow \mathbf{D}_b$ } (i.e. the set of all functions f from $\mathbf{D}_a$ to $\mathbf{D}_b$.)

$\mathbf{D}_{<s,a>}$ = {f | f: W → $\mathbf{D}_a$ } (i.e. the set of all functions f from W to $\mathbf{D}_a$.)

The semantic interpretation of IL also makes use of a set G of assignment functions *g*, functions from variables of all types to values in the corresponding domains.

Each expression of IL has an *intension* and, at each *w* in W, an *extension*. The intension is relative to **M** and *g*; the extension is relative to **M**, *w,* and *g*. But we will not discuss intensions and extensions in this lecture.

A.2.2. Atomic expressions ("lexicon"), notation, and interpretation.

The atomic expressions of IL are constants and variables; there are infinitely many constants and infinitely many variables in each type. Montague introduced a general nomenclature for constants and variables of a given type, using *c* and *v* with complex subscripts indicating type and an index. In practice, including Montague's, more mnemonic names are used. Our conventions will be as follows:

Constants of IL will be written in **non-italic boldface**, and their names will usually reflect the English expressions of which they are translations: **man, love**, etc. Their types will be specified. Variables of IL will be written in ***italic boldface***, usually observing the following conventions as to types:

Type e: ***w,x,y,z,*** with and without subscripts or primes (this modifier holds for all types.)

Type <e,t>: ***P, Q***

Various relational types such as <e,<e,t>>: ***R***

The type of generalized quantifiers: ***T***

The interpretation of constants is given by the interpretation function I of the model, and the interpretation of variables by an assignment *g*, as specified in Rule 1 below.

A.2.3. Syntactic rules and their model-theoretic semantic interpretation

The syntax of IL takes the form of a recursive definition of the set of "meaningful expressions of type *a*", ME_a, for all types *a*. The semantics gives an interpretation rule for each syntactic rule.

Note: when giving syntactic and semantic rules for IL, as for predicate logic, we use a metalanguage which is very similar to IL; but we are not boldfacing the constants and variables of the metalanguage. The metalanguage variables over variables are most often chosen as *u* or *v*.

The first rule is a rule for atomic expressions, and the first semantic rule is its interpretation:

Syntactic Rule 1: Every constant and variable of type *a* is in ME_a.
Semantic Rule 1: (a) If α is a constant, then $\|\alpha\|^{M,w,g} = I(\alpha)(w)$.
(b) If α is a variable, then $\|\alpha\|^{M,w,g} = g(\alpha)$.

Note: The recursive semantic rules give *extensions* relative to model, world, and assignment. Read "$\|\alpha\|^{M,w,g}$ " as "the semantic value (extension) of alpha relative to M, *w,* and *g*." The interpretation function I assigns to each constant an *intension*, i.e. a function from possible worlds to extensions; applying that function to a given world *w* gives the extension.

Syntactic Rule 2. (really a <u>set</u> of rules for logical connectives and operators that apply to formulas, mostly from propositional and predicate logic, plus some modal and tense operators.)
If φ, ψ ME_t, and *u* is a variable of any type, then ¬φ, φ&ψ, φ∨ψ, φ→ψ, φ↔ψ (also written as φ≡ψ), ∃*u*φ, ∀*u*φ, □φ, **PAST**φ ∈ ME_t. Note: " □φ" is read as "Necessarily phi".

Semantic Rule 2:
(a) ¬φ, φ&ψ, φ∨ψ, φ→ψ, φ↔ψ, ∃*u*φ, ∀*u*φ as in predicate logic.
(b) $\|\Box\varphi\|^{M,w,g} = 1$ iff $\|\varphi\|^{M,w',g} = 1$ for all *w'* in W.
(c) $\|$ **PAST** $\varphi\|^{M,w,g} = 1$ iff $\|\varphi\|^{M,w',g} = 1$ for some $w' \leq w$. (This is a simplification; here we are treating each *w* as a combined "world/time index", possibly a situation index; $w' \leq w$ if *w'* is a temporally earlier slice of the same world as *w*.)

Syntactic Rule 3: (=): If $\alpha,\beta \in ME_a$, then $\alpha=\beta \in ME_t$.
Semantic Rule 3: $\|\alpha=\beta\|^{M,w,g} = 1$ iff $\|\alpha\|^{M,w,g} = \|\beta\|^{M,w,g}$.

(The next two pairs of rules, concerning the "up" and "down" operators, are crucial for intensionality, but we will not discuss them and will not use them.)

Syntactic Rule 4: ("up"-operator.) If $\alpha \in ME_a$, then $[^\wedge\alpha] \in ME_{<s,a>}$.
Semantic Rule 4: $\| [^\wedge\alpha]\|^{M,w,g}$ is that function *h* of type <*s,a*> such that for any *w'* in W, $h(w') = \|\alpha\|^{M,w',g}$.
Syntactic Rule 5: ("down"-operator.) If $\alpha \in ME_{<s,a>}$, then $[^\vee\alpha] \in ME_a$.
Semantic Rule 5: $\| [^\vee\alpha]\|^{M,w,g}$ is $\|\alpha\|^{M,w,g}(w)$

The next two pairs of rules, function-argument application and lambda-abstraction, are among the most important devices of IL, and we will make repeated use of them.

<u>Function-argument application:</u>
Syntactic Rule 6: If $\alpha \in ME_{<a,b>}$ and $\beta \in ME_a$, then $\alpha(\beta) \in ME_b$.
Semantic Rule 6: $\|\alpha(\beta)\|^{M,w,g} = \|\alpha\|^{M,w,g}(\|\beta\|^{M,w,g})$

<u>Lambda-abstraction:</u>
Syntactic Rule 7: If $\alpha \in ME_a$ and *u* is a variable of type *b*, then $\lambda u[\alpha] \in ME_{<b,a>}$.
Semantic Rule 7: $\| \lambda u[\alpha] \|^{M,w,g}$ is that function *f* of type $b \rightarrow a$ such that for any object d of type *b*, $f(d) = \|\alpha\|^{M,w,g[d/u]}$.

==========

HOMEWORK #1, Due at the time of Lecture 4.

Do at least problems 1 and 2. 2 pages should be enough; 4 pages maximum. There is an extra page of "homework help" for this homework. First try to do it without looking at the help, then look at the help, and then try it again if you did it wrong the first time. Bring questions to seminar next week if you would like additional help then. Optional extra problems: 3-8. The last three of the <u>extra (optional)</u> problems require next week's handout as background.
Note: Don't forget to bring or e-mail to me Homework #0, the "Anketa"!

==========

1. (a) Write down the translation into the λ-calculus of "A student walks and talks". This handout already shows the translations of "a student" and "walks and talks". Put them together by "function-argument application".
(b) Apply λ-conversion to simplify the formula. There will be two applications, and the resulting formula should have no λ's.
(c) Write down the translation of "A student walks and a student talks"; simplify by λ-conversion.
(d) The two formulas (if you did parts (a-c) correctly) are not equivalent. Describe a situation (a model) in which one of them is true and the other one is false.

2. In the predicate calculus, the sentence "No student talks" can be represented as follows:
¬∃***x*** [**student**(***x***) & **talk**(***x***)] or equivalently as ∀***x***[**student**(***x***) → ¬**talk**(***x***)]
But in the predicate calculus, there is no way to represent the meaning of the NP "no student". Using the λ-calculus in the way illustrated above for the NPs "every student", "a student", "the king", write down a translation for the NP "no student". (There are two logically equivalent correct answers; write down either or both.)

Additional questions, optional but recommended. These will be easier after Lecture 3.

3. Write out the semantic derivation of *John walks* two ways, once using Montague's generalized quantifier interpretation of *John*, once using the type-e interpretation of *John*.

4. Write the translation for *every*, by abstracting on the CNP in the given translation of *every man*. Hint: the translation of *every*, like that of every DET, should begin with $\boldsymbol{\lambda Q}$, where $\boldsymbol{Q}$ is a variable of type e → t, the type of the CNP with which the DET "wants to" combine.

5. Work out the translation, using lambda-conversion for simplification of results, of the sentence:

Every violinist who loves Prokofiev is happy.

Begin by figuring out how the sentence was put together syntactically – draw the tree! --, so that you can see how to do the translation compositionally.

As you build up the translation, always apply lambda-conversion as soon as it is applicable, so that the formulas do not become more complex than necessary.

When you do homework, feel free to write questions on your homework paper.

==========

HELP FOR HOMEWORK #1

Help with Problem 1. Instead of giving an answer, here is an answer to a similar problem, which we'll call Problem 1*.

Problem 1*. (a) Write down the translation into the λ-calculus of "Every student walks and talks". The handout already shows the translations of *every student* and *walks and talks*. Just put them together by function-argument application.

(b) Apply λ-conversion to simplify the formula. There will be two applications, and the resulting formula should have no λ's.

(c) Write down the translation of "Every student walks and every student talks"; simplify by λ-conversion.

(d) Are the two formulas equivalent? Give an argument.

Answer.

(1*) (a) *every student* : $\lambda P[\forall x\ (\mathbf{student}(x) \rightarrow P(x))]$

walks and talks: $\lambda y[(\mathbf{walk}\ (y)\ \&\ \mathbf{talk}(y))]$

every student walks and talks: $\lambda P[\forall x\ (\mathbf{student}(x) \rightarrow P(x))]\ (\lambda y[(\mathbf{walk}\ (y)\ \&\ \mathbf{talk}(y))])$

(b) Simplify the expression by two applications of λ-conversion.

Step 1: $\boldsymbol{\lambda P[\forall x\ (\mathbf{student}(x) \rightarrow \mathbf{P}(x))]\ (\lambda y[(\mathbf{Walk\ (y)\ \&\ Talk(y))])} \equiv$
$\forall x\ (\mathbf{student}(x) \rightarrow \boldsymbol{\lambda y[(\mathbf{Walk\ (y)\ \&\ Talk(y))}]\ (x))}$

Step 2: $\forall x\ (\mathbf{student}(x) \rightarrow \boldsymbol{\lambda y}[(\mathbf{walk}\ (y)\ \&\ \mathbf{talk}(y))]\ (x)) \equiv$
$\forall x\ (\mathbf{student}(x) \rightarrow (\mathbf{walk}\ (x)\ \&\ \mathbf{talk}(x)))$

(c) *Every student walks and every student talks*:
$\lambda P[\forall x\ (\mathbf{student}(x) \rightarrow P(x))]\ (\mathbf{walk})\ \&\ \lambda P[\forall x\ (\mathbf{student}(x) \rightarrow P(x))]\ (\mathbf{talk})$
$\equiv \forall x\ (\mathbf{student}(x) \rightarrow \mathbf{walk}(x))\ \&\ \forall x\ (\mathbf{student}(x) \rightarrow \mathbf{talk}(x))$
It doesn't matter whether the same variable x is used in both formulas or not; this is equivalent to: $\forall x\ (\mathbf{student}(x) \rightarrow \mathbf{walk}(x))\ \&\ \forall y\ (\mathbf{student}(y) \rightarrow \mathbf{talk}(y))$

(d) Use first-order predicate logic to argue that the last formula in (b) and the last formulas in (c) are equivalent. Both formulas require that every student have both properties.

Note: If two formulas ARE equivalent, you should use what you know about predicate logic to try to prove the equivalence (either a formal proof or an informal argument). If two formulas are NOT equivalent, then you should construct a model (often a very small model is enough) in which one of the formulas is true and the other one is false: that's always the best and simplest way to show non-equivalence.