Lecture 2. Lambda abstraction, NP semantics, and a Fragment of English

NOTE: I know this is too much for one lecture. But it is useful to have all of this in one handout. We will spend this time and part of next time on this, with more next time about the semantics of noun phrases as Generalized Quantifiers.

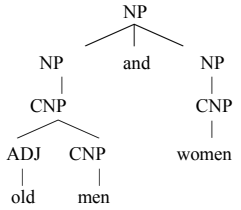
1. Lexical and Structural Ambiguity

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

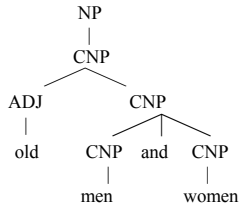
Structural ambiguity. Compositionality requires a "disambiguated language" (a "language without ambiguity"). So we interpret expressions ***with syntactic structure,*** not just strings.

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

(a)

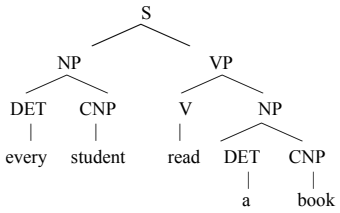


(b)



(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) $\rightarrow$ $\rightarrow$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.

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 φ ∈ Form and v is a variable, then λv[φ] ∈ Pred-1.

S9: $\rangle\lambda v[\varphi]^{M, g}$ is the set S of all d 0 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) $\rangle\| \boldsymbol{\lambda x}[\mathbf{run}(\boldsymbol{x})] \|^{M, g}$ = the set of all individuals that run.

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

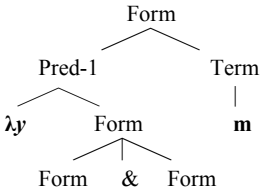
(iii) $\rangle\| \boldsymbol{\lambda x}[\mathbf{love}(\boldsymbol{x}, \boldsymbol{y})] \|^{M, g}$ = the set of all individuals that love $\| \boldsymbol{y} \|^{M, g[d/x]}$, i.e. g(***y***), i.e. John.

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

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

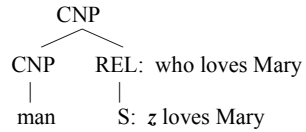
(vi) to represent "Mary walks and talks" with constituents that correspond to surface syntax:
$\boldsymbol{\lambda y}[(\mathbf{walk}(\boldsymbol{y}) \,\&\, \mathbf{talk}(\boldsymbol{y}))]\,(\mathbf{m})$

Syntactic Structure of the formula in (vi):



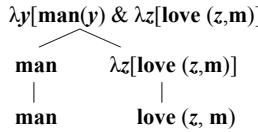
By λ-conversion (see below), the formula in (vi) is equivalent to (**walk**(**m**) & **talk**(**m**)).

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



Rule for combining CNP and REL: λy[CNP'(***y***) & REL'(***y***)] (combining "translations")

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



By λ-conversion (see below), the top line is equivalent to: λy[**man**(***y***) & **love** (***y***,**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 the 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: $\lambda x[x^2 + 1]$ denotes the function $x \rightarrow x^2 + 1$.
Function-argument application: $\lambda x[x^2 + 1]\,(5) = 26$

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

Lambda-conversion Rule: $\lambda v[\alpha](\beta) \equiv \alpha'$, 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].)

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

4. English Fragment 1.

4.0 Introduction

In this part we present a small sample English grammar (a "fragment", in MG terminology), that is, an explicit description of the syntax and semantics of a small part of English. This fragment is intended to serve several purposes: making certain aspects of formal semantics more explicit, including (and illustrating) more of the basics of the lambda-calculus. The fragment is of interest in its own right and will also serve as background for the next lecture. The fragment, with its very minimal lexicon, also illustrates the typically minimal treatment of the lexicon in classical Montague grammar.

The semantics of the fragment will be given via translation into Montague's Intensional Logic (IL) (the alternatives would be to give a direct model-theoretic interpretation, or an interpretation via translation into some other model-theoretically interpreted intermediate language). In the first part of this lecture we presented Montague's IL: Its type structure and the model structures in which it is interpreted, and its syntax and model-theoretic semantics.

Now we introduce the fragment of English: first the syntactic categories and the category-type correspondence, then the basic syntactic rules and the principles of semantic interpretation, and then a small lexicon and some meaning postulates. In Section 5 we present some examples. Certain rules of the fragment are postponed to Section 6 where they receive separate discussion; these are rules that go beyond the simple phrase structure rule schemata of Section 4. (We will do some of this today, and the rest next time: bring this handout next time too!)

4.1. Syntactic categories and their semantic types.

Syntactic category	Semantic type (extensionalized)	Expressions
ProperN	e	names (*John*)
S	t	sentences
CN(P)	e → t	common noun phrases (*cat*)
NP	(i) e	"e-type" or "referential" NPs (*John, the king*)
	(ii)(e→ t) → t	noun phrases as generalized quantifiers (*every man, the king, a man, John*)
	(iii) e → t	NPs as predicates (*a man, the king*)
ADJ(P)	(i) e → t	predicative adjectives (*carnivorous, happy*)
	(ii)(e → t) → (e → t)	adjectives as predicate modifiers (*skillful*)
REL	e → t	relative clauses (*who(m) Mary loves*)
VP, IV	e → t	verb phrases, intransitive verbs (*loves Mary, is tall, walks*)
TV(P)	type(NP) → type(VP)	transitive verb (phrase) (*loves*)
is	(e → t) → (e → t)	*is*
DET	type(CN) → type(NP)	*a, some, the, every, no*

4.2. Syntactic Rules and Semantic Rules.

Two different approaches to semantic interpretation of natural language syntax (both compositional, both formalized, and illustrated, by Montague):

A. Direct Model-theoretic interpretation: Semantic values of natural language expressions (or their "underlying structure" counterparts) are given directly in model-theoretic terms; no intermediate language like Montague's intensional logic (but for some linguists there is a syntactic level of "logical form" to which this model-theoretic interpretation applies, so the distinction between the two strategies is not always sharp.) This is the direct "English as a formal language" strategy. For illustration, see Heim and Kratzer (1998). Also see the discussion in Larson's chapter 12.

B. Interpretation via translation: Stage 1: compositional translation from natural language to a language of semantic representation, such as Montague's intensional logic. For an expression, of category C formed from expressions, of category A and, of category B, determine **TR**(γ) as a function of **TR**(α) and **TR**(β). Stage 2: Apply the compositional model-theoretic interpretation rules to the intermediate language.

We will follow the strategy of interpretation via translation, using Montague's IL as the intermediate language. But everything we do could also be done by direct interpretation.

Some abbreviations and notational conventions:

We will sometimes write α' as a shorthand for **TR**(α). And sometimes we use the category name in place of a variable over expressions of that category, writing **TR**(A), or A', in place of **TR**(α) when α is an expression of category A. And we will write some of our syntactic rules like simple phrase structure rules. Here is an example of a syntactic rule and corresponding translation rule, and their abbreviations as they will appear below.

Official Syntactic Rule: If α is an expression of category DET and β is an expression of category CNP, then $F_0(\alpha,\beta)$ is an expression of category NP, where $F_0(\alpha,\beta) = \alpha\beta$.
Official Semantic Rule: If **TR**(α) = α' and **TR**(β) = β', then **TR**($F_0(\alpha,\beta)$) = α'(β').

Abbreviated Syntactic Rule: NP → DET CNP
Abbreviated Semantic Rule: NP' = DET'(CNP')

4.2.1. Basic syntactic rules

Basic rules, phrasal:

S → NP VP
NP → DET CNP
CNP → ADJP CNP
CNP → CNP REL
VP → TVP NP
VP → is ADJP
VP → is NP

Basic rules, non-branching rules introducing lexical categories:

NP → ProperN
CNP → CN
TVP → TV
ADJP → ADJ
VP → IV

4.2.2. Semantic interpretation of the basic rules.

The basic principle for all semantic interpretation in formal semantics is the principle of compositionality; the meaning of the whole must be a function of the meanings of the parts. In the most "stipulative" case, we write a semantic interpretation rule (translation or direct model-theoretic interpretation) for each syntactic formation rule, as in classical MG. In more contemporary approaches, we look for general principles governing the form of the rules and their correspondence (possibly mediated by some syntactic level of "Logical Form".) Here we are using an artificially simple fragment, and we have presented the syntactic rules in a form which is explicit but not particularly general; but we have the tools to illustrate a few basic generalizations concerning syntax-semantics correspondence.

4.2.2.1. Type-driven translation. (Partee 1976, Partee and Rooth 1983, Klein and Sag 1985)

To a great extent, possibly completely, we can formulate general principles for the interpretation of the basic syntactic constructions based on the semantic types of the constituent parts.

So suppose we are given a rule A → B C, and we want to know how to determine A' as a function of B'and C' (equivalently, **TR**(A) as a function of **TR**(B) and **TR**(C); and ultimately, ||A|| as a function of ||B|| and ||C||.) Similarly for non-branching rules A → B.

General principles: function-argument application, predicate conjunction, identity.
The following versions of general type-driven interpretation principles are taken from Heim and Kratzer (1995).
In the original they are written for direct model-theoretic interpretation.
(1) *Terminal Nodes* (TN): If α is a terminal node, then [[α]] is specified in the lexicon.

(2) *Non-Branching Nodes* (NN): If α is a non-branching node, and β is its daughter node, then [[α]]œ = [[β]].
(3) *Functional Application* (FA): If α is a branching node, {β,γ} is the set of α's daughters, and [[β]] is a *function* whose domain contains [[γ]], then [[α]] = [[β]] ([[γ]]).
(4) *Predicate Modification* (PM) : If α is a branching node, {β, γ} is the set of α's daughters, and [[β]] and [[γ]] are both in $D_{<e,t>}$, then [[α]] = $\lambda x \in D_e$. [[β]] (x) = 1 and [[γ]] (x) = 1.

(A further principle is needed for *intensional functional application*, which we will mention only later.)
Exactly analogous principles can be written for **type-driven *translation***:

(1) *Terminal Nodes* (TN): If α is a terminal node, then **TR**(A) is specified in the lexicon.
(2) *Non-Branching Nodes* (*NN*): If A → B is a unary rule and A,B are of the same type, then **TR**(A) = **TR**(B).
(3) *Functional Application* (FA): If A is a branching node, {B,C} is the set of A's daughters, and B' is of a functional type a → b and C' is of type a, then **TR**(A) = **TR**(B)(**TR**(C)).
(4) *Predicate Modification* (*PM*) : If A is a branching node, {B,C} is the set of A's daughters, and if B' and C' are of (same) predicative type a → t, and the syntactic category A can also correspond to type a → t, then **TR**(A) = λx[**TR**(B)(x) & **TR**(C)(x)]. (i.e. ||A|| = ||B|| ∩ ||C||.)

4.2.2.2. Result of those principles for the translation of the basic rules.

Function-argument application: S→ NP VP, NP → DET CNP, VP → TVP NP, VP → *is* ADJP, VP → *is* NP, and those instances of CNP → ADJP CNP in which ADJP is of type (e→t)→(e→t).

Example: Consider the rule S→NP VP. If NP is of type (e→t)→t and VP is of type e→t, then the translation of S will be NP'(VP') (e.g., *Every man walks*). If NP is of type e and VP is of type e→t, then the translation of S will be VP'(NP') (e.g., *John walks*).

Predicate modification: CNP → CNP REL, and those instances of CNP → ADJP CNP in which ADJP is of type e→t.

Non-branching nodes: NP → ProperN, CNP → CN, TVP → TV, ADJP → ADJ.

4.2.3. Rules of Relative clauses, Quantification, Phrasal Negation. See Section 6.

4.2.4. Type multiplicity and type shifting.

We noted in Lecture 1 that classical model-theoretic semantics in the Montague tradition requires that there be a single semantic type for each syntactic category. But in Fragment 1, several syntactic types have more than one corresponding semantic type. The possibility of type multiplicity and type shifting has been increasingly recognized in the last decade or so, and there are a variety of formal approaches that accommodate type multiplicity without giving up compositionality. We will not go into details about formal issues here, but we do want to include a number of categories with multiple semantic types; several were introduced in Fragment 1, and more will be introduced in later lectures.

Montague tradition: uniform treatment of NP's as generalized quantifiers, type (e → t) → t.

John	λ***P***[***P***(**John**)]	(the set of all of John's properties)
a fool	λ***P***∃***x***[**fool**(***x***) & ***P***(***x***)]	
every man	λ***P***∀***x***[**man**(***x***) → **P**(***x***)]	

Intuitive type multiplicity of NP's (and see Heim 1982, Kamp 1981):

John	"referential use":	**John**	type e
a fool	"predicative use":	**fool**	type e → t
every man	"quantifier use":	(above)	type (e→t)→t

Resolution: All NP's have meaning of type (e→t)→t; some also have meanings of types e and/or e → t. General principles for predicting (Partee 1987). Predicates may semantically take arguments of type e, e → t, or (e→t)→t, among others.[1]

Type choice determined by a combination of factors including coercion by demands of predicates, "try simplest types first" strategy, and default preferences of particular determiners.

Note the effects of this type multiplicity on type-driven translation. The S → NP VP rule, for instance, will have two different translations. The VP, we have assumed, is always of type e → t. If the NP is of type e, the translation will be VP'(NP'), whereas if the NP is of type (e→t)→t, the translation will be NP'(VP'), as noted above in Section 4.2.2.2. **[See Homework problem #3.]**

4.3. Lexicon.

Here we illustrate the treatment of the lexicon in Montague (1973) ("PTQ"). Montague, not unreasonably, saw a great difference between the study of the principles of compositional semantics, which are very similar to the principles of compositional semantics for logical languages as studied in logic and model theory, and the study of lexical semantics, which he perceived as much more "empirical". For Montague, it was important to figure out the difference in logical type between *easy* and *eager*, or between *seem* and *try*, but he did not try to say anything about the difference in meaning between two elements with the same "structural" or type-theoretic behavior, such as *easy* and *difficult* or *run* and *walk*. For Montague, most lexical items were considered atomic expressions of a given type, and simply translated into constants of IL of the given type.

First we simply list some lexical items of various syntactic categories; aside from the category DET, these are all open classes. Then we discuss their semantics.

In later lectures we will be concerned with how best to enrich the semantic information associated with the lexicon in ways compatible with a compositional semantics.

ProperN: *John, Mary, Bill, ...*
DET: *some, a, the, every*
ADJ: *carnivorous, happy, skillful, tall, former, alleged, old, ...*
CN: *man, king, violinist, surgeon, fish, senator, ...*
TV: *sees, loves, catches, eats, ...*
IV: *walks, talks, runs, ...*

[1] (More on type-shifting in RGGU 2005 Lecture 8 (or see RGGU 2004 Lecture 6 on my website), and more on type multiplicity of adjectives in MGU Lecture 4.)

Semantics of Lexicon (MG):
Open class lexical items (nouns, adjectives, verbs) translated into constants of appropriate type (notation: English expressions *man, tall* translated into IL constants **man**, **tall**, etc.). Interpretation of these constants a central task of lexical semantics. A few open class words (e.g. *be*, *entity*, *former*) sometimes treated as part of the "logical vocabulary".

Closed class lexical items: some treated like open class items (e.g. most prepositions), others (esp. "logical" words) given explicit interpretations, as illustrated below.

Determiners:
We have three types of NPs and correspondingly three types of DETs. Not all DETs occur in all types; *the* is one of the few that does. For DETs that occur in more than one type, we will subscript the "homonyms" with mnemonic subscripts: **e** for those that combine with a CNP to form an e-type NP, **pred** for those that form predicate nominals, and **GQ** for those that form generalized quantifier-type NPs. (Note that these are not the types of the DETs themselves, but their own types have unpleasantly long and hard-to-read names.) There are systematic relations among these "homonyms" (Partee 1987), but we are not discussing them here.

(i) e-forming DETs.

For the translation of the_e, we need to add the *iota-operator* ι to IL.

Syntax: If φ ∈ ME_t and ***u*** is a variable of type *e*, then **ι*u***[φ] ∈ ME_e.

Semantics: $||\iota u[\varphi]||^{M,w,g}$ = d iff there is one and only one d ∈ D such that $||\varphi||^{M,w,g[d/u]}$ =1.
$||\iota u[\varphi]||^{M,w,g}$ is undefined otherwise.

So **ιx(king(*x*))** denotes the unique individual who is king, if there is one, and is undefined if there is either no king or more than one king.

The type for determiners as functors forming e-type ("referential") terms is (e → t) → e; the only determiner of this type we will introduce is the_e. For ease of reading, we will give the translations of representative NPs, rather than for the DET itself; the DET translation can be formed in each case by λ-abstraction on the CNP (see below, **TR**(a_{pred})).

TR(the_e *king*) = **ιx(king(*x*))**

(ii) predicate-forming DETs.
DETs as functors forming predicate nominals are of type (e → t) → (e → t).

TR(a_{pred} *man*) = **man**
TR(the_{pred} *man*) = **λx[man(*x*) & ∀*y*[man(*y*) → *y*=*x*)]]**

We illustrate the translation of the DET itself with the translation of a_{pred}.

TR(a_{pred}) = **λ*P*[*P*]**

(iii) generalized quantifier-forming DETs.
DETs as functors forming generalized quantifiers are of type (e→t)→((e→t)→t)

TR(a_{GQ} *man*) = **λ*P*∃*x*[man(*x*) & *P*(*x*)]**
TR($every_{GQ}$ *man*) = **λ*P*∀*x*[man(*x*) → *P*(*x*)] (see Homework problem 4)**
TR(the_{GQ} *man*) = **λ*P*∃*x*[man(*x*) & ∀*y*[man(*y*) → *y*=*x*)]& *P*(*x*)]**

The copula *be*:

TR(*is*) = **λ*P*λ*x*[*P*(*x*)]** ("Predicate!")

Results (see also Section 5, and Homework problems 7, 8):

TR(*is green*) = **green**
TR(*is* a_{pred} *man*) = **man**
TR(*is* the_{pred} *king*) = λx**[king(*x*) & $\forall y$[king(*y*) → *y*=*x*)]]**

5. Examples

(1) *is happy*

TR(*is*) = $\lambda P \lambda x$**[*P*(*x*)]**
TR(*happy*) = **happy**
TR(*is happy*) = $\lambda P \lambda x$**[*P*(*x*)] (happy)**
= λx**[happy(*x*)] = happy**

(2) *The violinist is happy* (with e-type interpretation of subject)

TR([$_{NP}$ *the violinist*]) = ιx**[violinist(*x*)]** type: e
TR([$_{VP}$ *is happy*]) = **happy** type: e → t
TR([$_{S}$ *the violinist is happy*] = **happy(ιx[violinist(*x*)])** type: t
(VP meaning applies to NP meaning)

(3) *Every violinist is happy* (with GQ-type subject)

TR(*every violinist*) = $\lambda P \forall x$**[violinist (*x*) → *P*(*x*)]** type (e → t) → t
TR(*is happy*) = **happy** type e → t
TR(*every violinist is happy*) = $\lambda P \forall x$**[violinist (*x*) → *P*(*x*)](happy)**
= $\forall x$**[violinist (*x*) → happy(*x*)]**

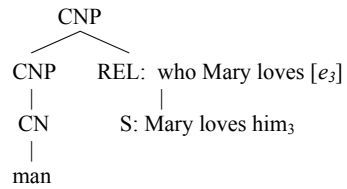
(4) *Every surgeon is a skillful violinist*
(The type of *every surgeon* must be (e → t) → t; the type of *a skillful violinist* must be e → t. Assume the type of *skillful* is (e → t) →(e → t).)
TR(*every surgeon*) = $\lambda P \forall x$**[surgeon (*x*) → *P*(*x*)]**
TR(*skillful violinist*) = **skillful(violinist)**
TR(*is a skillful violinist*) = **TR** (*a skillful violinist*)= **TR**(*skillful violinist*)
TR(*every surgeon is a skillful violinist*) = $\forall x$**[surgeon(*x*) → skillful(violinist)(*x*)]**

6. Rules of Relative clause formation, Quantifying In, Phrasal Negation.

6.1. (Restrictive) Relative clause formation.

We begin with an illustration of what the rule does before stating it (in a sketchy form). Consider the CNP *man who Mary loves*:

Syntactic derivation (very sketchy):



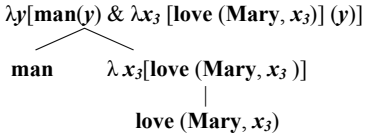
The types for CN, CNP, and REL are all e → t; so the principle for combining CNP and REL gives: λy[CNP'(*y*) & REL'(*y*)] (Predicate conjunction)

The relative clause itself is a predicate formed by λ-abstraction on the variable corresponding to the WH-word. (Partee 1976 suggests a general principle that all "unbounded movement rules" are interpreted as involving variable-binding; and λ-abstraction can be taken as the most basic variable-binding operation.)

A syntactically very crude and informal version of the relative clause rule, with its semantic interpretation, can be stated as follows:

Rel Clause Rule, syntax: If φ is an S and φ contains an indexed pronoun he_i / him_i in relativizable position, then the result of adjoining *who(m)* to S and leaving a trace e_i in place of he_i / him_i is a REL.
Rel Clause Rule, semantics: If φ translates as φ', then REL translates as λx_i[φ'].

Semantic derivation corresponding to the syntactic derivation above; compositional translation into IL: (read bottom-to-top) **(and see Homework problem 5a)**



By λ-conversion, the top line is equivalent to: λy[**man**(*y*) & **love** (**Mary**, *y*)]

6.2. Quantifying In.

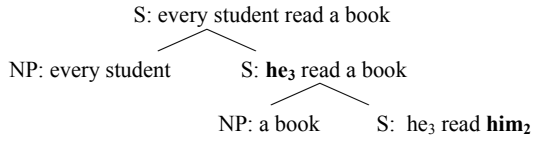
This is (an informal statement of) Montague's Quantifying In rule; it is similar to the Quantifier-Lowering rule of Generative Semantics and Quantifier Raising (QR) of May (1977); various alternative treatments of quantifier scope ambiguity exist, including Cooper-storage (Cooper 1975) and Herman Hendriks's flexible typing approach (Hendriks 1988, 1993).

Quantifying In Rule, Syntax: (informally stated): An NP combines with a sentence with respect to a choice of variable ("he_i" in MG). Substitute the NP for the first occurrence of the variable; change any further occurrences of the variable into pronouns of the appropriate number and gender.

Semantic rule: NP'(λx_i [S']) (The set of properties denoted by the NP includes the property denoted by the λ-expression derived from the sentence.)

We illustrate with two derivations for the ambiguous sentence *Every student read a book.*

Syntactic derivation (i) (rough sketch; read from bottom to top. **Bold** is used here to show which variables are substituted for at each step.)

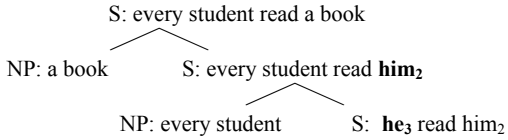


Compositional Translation: (every student)'(λx_3 [(a book)'(λx_2 [read (x_3 , x_2)])])

Rough paraphrase: Every student has the property that there is a book that he read.

If you write out the interpretations of the NPs and apply Lambda-Conversion as many times as possible, the result will be (some alphabetic variant of) the first-order PC formula ∀x(**student**(x) → ∃y(**book**(y) & **read**(x,y))).

Syntactic derivation (ii)



Compositional Translation: (a book)'(λx_2[(every student)'(λx_3 [read (x_3 , x_2)])]) [**See Homework problem 6**.]
Paraphrase: Some book has the property that every student read it.
After applying Lambda-Conversion as many times as possible, the result will be (some alphabetic variant of) the first-order PC formula ∃y(**book**(y) & ∀x(**student**(x)→ **read**(x,y))).

Observation: Compositional semantics requires that every ambiguous sentence be explainable on the basis of ambiguous lexical items and/or multiple syntactic derivations. Semantic structure mirrors syntactic part-whole structure, which in Montague Grammar is represented by syntactic derivational structure, not surface structure. There are different theories of the semantically relevant syntactic structure: "Derivation trees" or "analysis trees" (MG), LF (Chomskian GB or Minimalist theory), Tectogrammatic Dependency Trees (Prague), Deep Syntactic Structure (Mel'čuk) Underlying Structure (Generative Semantics), ... GPSG, HPSG, and various contemporary versions of Categorial Grammar are attempts to represent all the necessary syntactic information directly in a single "level" of syntax.

6.3. Conjunction.

One simple and elegant application of lambda abstraction which Montague used in PTQ is its use in defining the interpretation of "Boolean" phrasal conjunction, disjunction, and negation in terms of the corresponding sentential operations.

"Boolean" phrasal conjunction, illustrated in all the examples below, is distinguished from "part-whole" or "group" conjunction, illustrated by "John and Mary are a happy couple" and "The flag is red and white", which are not equivalent, respectively, to "John is a happy couple and Mary is a happy couple" and "The flag is red and the flag is white".

To illustrate this application, we add a few syntactic and semantic rules to our fragment. Note: in the semantic rules, we use S_1 and S_2, etc., to refer to the first and second S in the syntactic rule.

Syntactic rules for conjunction:	**Corresponding semantic rules:**
S → S and S	S' = S_1' & S_2'
S → S or S	S' = S_1' ∨ S_2'
VP → VP and VP	VP' = λx [VP_1'(x)& VP_2'(x)]
VP → VP or VP	VP' = λ x [VP_1'(x) ∨ VP_2'(x)]
NP → NP and NP	NP' = λP[NP_1'(P) & NP_2'(P)]
NP → NP or NP	NP' = λ P [NP_1'(P) ∨ NP_2'(P)]

The NP conjunction and disjunction rules presuppose that the NPs are interpreted as generalized quantifiers, type <<e,t>,t>; ***P*** is a variable of type <e,t>. (Conjoined NPs of type e can be interpreted as groups, but not as conjoined by Boolean conjunction.)

Examples:
Some animals swim and some animals fly. (S-conjunction)
Some animals swim and fly. (VP-conjunction)
Every fish and some birds swim. (NP-conjunction)
Every painting and every statue was photographed or (*was*) *videotaped.* (NP-conjunction and VP-conjunction (or conjunction of participles, if we omit the second 'was', but it's equivalent to VP conjunction). The rules do correctly "predict" which conjunction has wider scope. (**Optional unlisted extra homework question: work out the last example.** Treat "was photographed" and "was videotaped" as simple 1-place predicates for this exercise.)

We could extend the rules above, and generalize them (as is done in Partee and Rooth 1983), so as to include further types of phrasal conjunction such as the following:
John bought and read a new book. (TV conjunction)
No number is even and odd. (Predicate ADJP conjunction)
Mary saw an old and interesting manuscript. (Pre-nominal ADJP conjunction.)

In fact, we do not have to "stipulate" the rules one-by-one as we have done above; it is possible to predict them in a general way from the *types* of the expressions being conjoined. But that goes beyond the scope of these lectures; see Partee and Rooth 1983.

6.4. Phrasal and lexical negation.

As an additional augmentation of our grammar which adds further illustration of the application of the lambda calculus, let us consider the relations among sentence negation, phrasal negation, and lexical (prefixal) negation.

The syntax of sentential negation in English is slightly complicated because of its interaction with the system of verbal auxiliaries, which we have not included in our simple grammar. Let us ignore the syntactic complexities here and work with a "Logical Form" grammar in which we have a simple phrase structure rule S □ NEG S. And let us add to the lexicon an element "NOT" of syntactic category NEG, of semantic type t □ t, whose translation into IL is simply ¬. Then the interpretation of NEG S, by function-argument application, will just be ¬S'.

Now what about phrasal negation, like *not every boy*, *not today, Mary but not John, not very intelligent, not love Bill*? Negation, like conjunction, is a "Boolean" operation, and it

is easy to define its interpretation with expressions of various types on the basis of its interpretation with sentences. For example:
Syntax: VP → NEG_{VP} VP
Semantics: Type: Since the type of VP is e→t, the type of NEG_{VP} must be (e → t)→(e → t).
TR(VP2) = **TR**(NEG_{VP})(**TR**(VP1)
= λ***P***[λ***x***[¬ ***P***(***x***)]](**TR**(VP1))
= λ***x***[¬VP1' (***x***)]

Similar rules for negation of other phrasal categories can be derived in a uniform way. (**See Homework Problem 5b, which asks for NEG_{NP} and optionally NEG_{DET}.**)

Both negation and conjunction (and disjunction) thus have natural extensions to a wide range of types (see Partee and Rooth 1983), with meanings systematically derivable from their basic meanings, which apply to sentences.

Lexical negation, as in *unhappy, impossible, unbroken, inedible*, can also be defined with the use of lambdas:

TR(*unhappy*) = λ***x***[¬**TR**(*happy*)(***x***)] = λ***x***[¬**happy**(***x***)]

==========

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 6 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 " is a constant, then $\|\alpha\|^{M,w,g} = I(\alpha)(w)$.
(b) If " 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. (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 $\neg\varphi$, $\varphi\&\psi$, $\varphi\vee\psi$, $\varphi\rightarrow\psi$, $\varphi\leftrightarrow\psi$ (also written as $\varphi\equiv\psi$), $\exists u\varphi$, $\forall u\varphi$, $\Box\varphi$, **PAST**$\varphi \in ME_t$. Note: " $\Box\varphi$" is read as "Necessarily phi".

Semantic Rule 2:
(a) $\neg\varphi$, $\varphi\&\psi$, $\varphi\vee\psi$, $\varphi\rightarrow\psi$, $\varphi\leftrightarrow\psi$, $\exists u\varphi$, $\forall u\varphi$ 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 $[^\vee\alpha]0\ ME_a$.
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.

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

Lambda-abstraction:
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*, $f(d) = \|\alpha\|^{M,w,g[d/u]}$.

==========

REFERENCES.

Heim, I. and A. Kratzer (1998). *Semantics in Generative Grammar*. Oxford:Blackwell

Klein, Ewan and Ivan Sag (1985), "Type-driven translation", *Linguistics and Philosophy* 8, 163-201.

Lewis, David (1970) "General semantics" *Synthese* 22, 18-67; reprinted in D.Davidson and G.Harman (eds.), *Semantics of Natural Language*. Dordrecht: Reidel (1972), 169-218. Also reprinted in Partee, Barbara H. ed. 1976. *Montague Grammar*. New York: Academic Press.

Montague, R. (1973) "The Proper Treatment of Quantification in Ordinary English," in K.J.J. Hintikka, J.M.E. Moravcsik, and P. Suppes, eds., *Approaches to Natural Language*, Reidel, Dordrecht, 221-242, reprinted in Montague (1974) 247-270. Also reprinted in Portner, Paul, and Partee, Barbara H. eds. 2002. *Formal Semantics: The Essential Readings*. Oxford: Blackwell Publishers.

Montague, Richard (1974) *Formal Philosophy: Selected Papers of Richard Montague.* Edited and with an introduction by Richmond Thomason, New Haven: Yale Univ. Press.

Partee, Barbara H. (1976) "Semantics and syntax: the search for constraints" in C.Rameh (ed.) *Georgetown University Round Table on Languages and Linguistics 1976*, Georgetown: Georgetown University School of Languages and Linguistics (99-110).

Partee, Barbara (1987) "Noun phrase interpretation and type-shifting principles", in J. Groenendijk, D. de Jongh, and M. Stokhof, eds., *Studies in Discourse Representation Theory and the Theory of Generalized Quantifiers*, GRASS 8, Foris, Dordrecht, 115-143. Reprinted in: Portner, Paul, and Partee, Barbara H. eds. 2002. *Formal Semantics: The Essential Readings*. Oxford: Blackwell Publishers.

Partee, Barbara and Mats Rooth, "Generalized conjunction and type ambiguity", in R. Bauerle, C. Schwarze, and A. von Stechow (eds.) *Meaning, Use and Interpretation of Language*, Walter de Gruyter, Berlin (1983) 361-383. Reprinted in: Portner, Paul, and Partee, Barbara H. eds. 2002. *Formal Semantics: The Essential Readings*. Oxford: Blackwell Publishers.

Partee, B., A. ter Meulen, and R.E. Wall (1990) *Mathematical Methods in Linguistics*, Dordrecht: Kluwer Academic Publishers.

Portner, Paul, and Partee, Barbara H. eds. 2002. *Formal Semantics: The Essential Readings*. Oxford: Blackwell Publishers.

HOMEWORK #1, Due March 8.

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, if you are able to do more: 3-8. Note: Don't forget to give 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:
$\neg\exists x\,[\mathbf{student}(x) \,\&\, \mathbf{talk}(x)]$ or equivalently as $\forall x[\mathbf{student}(x) \rightarrow \neg\mathbf{talk}(x)]$
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.

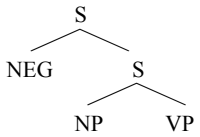
3. Write out the semantic derivation of *John walks* two ways, once using Montague's generalized quantifier interpretation of *John*, once using the type-e interpretation of *John*.

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 $e \rightarrow t$, the type of the CNP with which the DET "wants to" combine.

5. Work out the translations, using lambda-conversion for simplification of results, of the following. Always apply lambda-conversion as soon as it is applicable, so that the formulas do not become more complex than necessary.
(a) *Every violinist who loves Prokofiev is happy.*
(b) *Not every violinist is unhappy.*

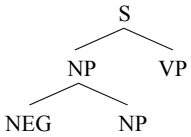
Note on exercise 5(b). (*Not every violinist is unhappy.*)
Note: Work this out two ways, which should come out equivalent. First pretend that the *not* is sentential negation, although according to the rules of English syntax, this is not a possible position for a sentential *not*. So the first syntactic structure should begin as follows:

(i)



Then figure out what the type and translation should be for a *not* which can apply to NPs of type $(e \rightarrow t) \rightarrow t$, and work out the translation for the sentence under an NP-negation analysis, where the syntactic structure begins as follows:

(ii)



There is a third possibility, which is to apply *not* to *every*; if you have figured out how to do (i) and (ii), you'll be able to figure out how to do the third; it's just more lambdas. Don't do it unless you really want to. What might be more interesting would be to work on linguistic arguments to try to decide how many of the three are real possibilities for English, and/or the same question for the corresponding Russian sentence.

6. Fill in the missing steps in the derivation of reduced forms of translations of the example *every student reads a book,* on derivation (ii) in section 6.2. Use Montague's generalized quantifier interpretations of *every student* and *a book*, as given in Section 3, repeated under "generalized quantifier-forming DETS" in Section 4.3. This is an exercise in compositional interpretation and lambda-conversion.

7. Fill in all the steps to show why $\mathbf{TR}(is\ a_{pred}\ man) = \mathbf{man}$. (End of section 4.3)

8. Write the translations of *the* in each of its three types. (Section 4.3)

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

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

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

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

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

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

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

(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 or 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.