# Lambda Abstraction
# (Chierchia & McConnell-Ginet Chapter 7)

**Lambda calculus was invented by Alonzo Church in the 1940s and 50s. In 1957 John McCarthy designed the programming language LISP, based on the lambda calculus. The Greek letter lambda is: λ**

**These days the lambda calculus is used in two different but related ways by linguists and linguistically-minded computer scientists**

- **to describe a logic of (logical) types, for programming languages and program execution**
- **to simplify/unify the description of the semantics for some linguistic constructions**

**(We'll just talk about aspects of the second type, with only a brief nod to issues concerned with type-shifting).**

**Problem:**

**a) *and* is a sentence connective in our PC and IPC semantic representation**

**b) *and* occurs as a VP-connective in English**

**For example: *Mary kissed George and drove home***

**STRATEGY 1:**

**Add new syntactic rule**

**$VP \rightarrow VP_1\ conj_{vp}\ VP_2$ ($conj_{vp}$ to distinguish from ordinary sentence conj's)**

**Add new raising rule**

**$[_{tp}\ NP\ [_{vp}\ VP_1\ conj_{vp}\ VP_2] \Rightarrow [_{tp}\ [_{tp}\ NP\ VP_1]\ conj\ [_{tp}\ NP\ VP_2]$**

**(conj is the ordinary sentence conjunction corresponding to $conj_{vp}$)**

**(we talk about *and*-conjunction first, but be aware that there are other types of VP-conjunction, such as *or*-conjunction; and possibly there is *if*-conjunction, *if and only if*-conjunction, *only*-conjunction, and *unless*-conjunction)**

**Problem with STRATEGY 1:**

**Sentences with quantified NPs in the subject:**

*A girl kissed George and drove home*
*Some girl kissed George and drove home*
*Most girls kissed George and drove home*
*A few girls kissed George and drove home*
**Etc.**

**The result of the raising rule would be:**

*A girl kissed George and a girl drove home*
*Some girl kissed George and some girl drove home*
*Most girls kissed George and most girls drove home*
*A few girls kissed George and a few girls drove home*

**(Also, researchers always had difficulty determining the exact conditions under which this sort of rule would apply).**

**So, STRATEGY 2:**

**Add new syntactic rule**

**$VP \rightarrow VP_1\ conj_{vp}\ VP_2$ ($conj_{vp}$ to distinguish from ordinary sentence conj's)**

**Try to find a semantic representation of this VP that could directly combine with the subject NP, while converting $conj_2$ into the right sentence connective.**

**λ-calculus to the rescue! Using the resources of λ-calculus, we can define properties that correspond to VPs (and to other things, coming up later) that will have the correct behavior and will be able to combine with the NP subject.**

**To explain this in general we start with the simpler cases of using λ-calculus, before even trying to do conjunctions. After that, we get to conjunctions, relative clauses, and VP ellipsis.**

**Take a "logical predicate", such as an intransitive verb (e.g., *smiled*) or *be+PredAdj* (e.g., *is happy*):**

**We can use l to form the "property expression" that corresponds to these:**

**λx[smiled(x)]  λx[happy(x)]**

**by "lambda abstraction". The item thereby formed is called a *lambda-abstract*, or a *lambda-term***

**The way to read these terms is:**

**"the property of (actually) having smiled"**

**"the property of (actually) being happy"**

**[or, "the property of being one of the things that smiled", "the property of being one of the happy things"]**

**(This somewhat stilted way of putting it is due to the fact that we are talking about the <u>extensional</u> case. After we move to the <u>intensional</u> case we can just say "the property of having smiled" and "the property of being happy".)**

**The very same thing can be done with any VP that has only one open "slot" (PRED$_1$, in the terminology of C&M-G):**

***kissed Mary*: λx[kissed(x,Mary)] "The property of actually having kissed Mary"**

***kissed someone*: λx[∃y(kissed(x,y)] "The property of actually having kissed someone"**

**Consider VP-conjunction now.**

**[$_{vp}$ *smiles and kissed Mary*] ⇒ λx[smiles(x) ∧ kissed(x,Mary)]**
**"the property of actually having smiled and kissed Mary"**

**Note that there is a sort of "type mismatch" between the *and*'s under discussion. The conj that we've always talked about joins sentences (which designate a truth value, in a model), where the $conj_{vp}$ joins VPs, which do *not* designate a truth value, in a model, but rather designate the *set of things* which manifest the feature that is designated by the VP. So what we want is to form the set of things which have manifested both VP-sets…i.e., their intersection. So $conj_{vp}$ means something like *set intersection*, but we can represent that by the sentence-conjunction of formulas…. Which is one of the advantages of λ-calculus.**

**Our authors do not especially remark on this, and say on p.409 (rule 32) that you just use the ordinary sentence conj to conjoin VPs. On the other hand, they do define what the new $conj_{vp}$ would look like:**

$$[P_1 \; conj_{vp} \; P_2] = \lambda x[P_1(x) \wedge P_2(x)]$$

**This is called a *point-wise definition* of $conj_{vp}$ because it defines $conj_{vp}$ in terms of what it does to each individual *x* in the domain, rather than to the $P_1$'s and $P_2$'s themselves.**

**Lambda-Conversion:**

**(Suppose part of our grammar is: NP→PN and S→NP VP)**

**And we have just described λx[smiled(x)] as the semantic representation of the intransitive VP *smiled*. And we might have *r* for the semantic representation of the PN *Ralph*. Then the sentence**

***Ralph smiled***

**ought to have that the VP-representation apply to the NP-representation:**

**(a) λx[smiled(x)] (*r*)**

**"*r* has the property of (actually) having smiled". We can move from the way this is stated in (a) to**

**(b) smiled(*r*)**

**by the process of lambda-conversion. All you do is….**

**Conceptually λ-conversion is pretty simple. You just replace the λ-bound variable with the argument. But you can get more tricky with complex formulas, and in these cases one needs to carefully check the scope of the λ-term.**

**λu∀x[λz[λw[ [Q(w) ∧ B(m,u)] ↔ K(x,w)](z)](j) → λy[K(y,x) ∨ Q(y)](m)] (a)**

**"inside-out": λw[ [Q(w) ∧ B(m,u)] ↔ K(x,w)](z) ⇒ [ [Q(z) ∧ B(m,u)] ↔ K(x,z)]**

**λz [ [Q(z) ∧ B(m,u)] ↔ K(x,z)] (j) ⇒ [ [Q(j) ∧ B(m,u)] ↔ K(x,j)]**

**λy[K(y,x) ∨ Q(y)](m) ⇒ [K(m,x) ∨ Q(m)]**

**This leaves us with**

**λu∀x [ [Q(j) ∧ B(m,u)] ↔ K(x,j)] → [K(m,x) ∨ Q(m)] ] (a) ⇒**

**∀x [ [Q(j) ∧ B(m,a)] ↔ K(x,j)] → [K(m,x) ∨ Q(m)] ]**

**Obviously you have to be careful with variable clashing between λ-terms and quantified terms.**

**Now go to the practice translation slides:**

***Every hobbit is short or is fat***

**vs.**

***Every hobbit is short or every hobbit is fat***

**Let's look at transitive verbs, such as *loves*. The representation of *loves* is:**

**λu[λv[loves(u)](v)] or for shorthand, λuλv[loves(u)(v)]**

**If we first apply this to *Sally*:**

**λuλv[loves(u)(v)] (*s*)**

**we get: λu[loves(*s*)(u)] i.e., "the property of loving Sally"**

**and if we now apply this to *George*:**

**we get: loves(*s*)(*g*). <Note order of arguments>**

**This is the general way transitive (and di-transitive) verbs work.**

**<Now go to the overheads for >**
***An elf knows Gandalf and loves Aragorn***
**vs.**
***An elf knows Gandalf and an elf loves Aragorn***

**Let's now consider conjoined $V_t$'s that are part of a VP with a "displaced" object:**

***An elf knows and loves Aragorn***

***John saw and bought a shirt***

**For two-place predicates: $[\text{Pred1 conj}_2 \text{ Pred2}] = \lambda x \lambda y[\text{Pred1}(x)(y) \wedge \text{Pred2}(x)(y)]$**
**The syntactic analysis of the VP yields a LF representation (something like):**

$[_{VP}\ [_{Vt}[_{Vt}\ \text{knows}]\ \text{and}\ [_{Vt}\ \text{loves}]\ ]\ [_{NP}\ \text{Aragorn}]\ ]$

**So we will get (for the embedded $V_t$)**

$\lambda v[\lambda u\ [\text{knows}(u)(v)]\ ]\ \text{conj}_2\ \lambda v[\lambda u\ [\text{loves}(u)(v)]\ ]$ **from whence (by above rule)**

$\lambda x \lambda y[\lambda v[\lambda u\ [\text{knows}(u)(v)]\ ](x)(y) \wedge \lambda v[\lambda u\ [\text{loves}(u)(v)]\ ](x)(y)\ ]\ ] \Rightarrow$

$\lambda x \lambda y\ [\text{knows}(x)(y) \wedge \text{loves}(x)(y)]$

**Now for the VP:**

$\lambda x \lambda y\ [\text{knows}(x)(y) \wedge \text{loves}(x)(y)]\ (\text{Aragorn}) \Rightarrow$

$\lambda y\ [\text{knows}(\text{Aragorn})(y) \wedge \text{loves}(\text{Aragorn})(y)]$

**But the VP is embedded in a TP with a NP-trace, say $t_1$ , and this yields**

$\lambda y$ [knows(Aragorn)(y) $\wedge$ loves(Aragorn)(y)] ($x_1$) $\Rightarrow$

[knows(Aragorn)($x_1$) $\wedge$ loves(Aragorn)($x_1$)]

**The subject is *An elf* (which is co-indexed with $x_1$)which combines with this to yield:**

$\exists x$(Elf($x_1$) $\wedge$ [knows(Aragorn)($x_1$) $\wedge$ loves(Aragorn)($x_1$)] )

**i.e., *Some elf knows and loves Aragorn***

**λ-abstraction/conversion in <u>intensional</u> logic.**

**Always before, when we said λ x[P(x)], this was the property of *actually* being P, or maybe "being P in the model M", or "being one of the P-things in M". Technically speaking this means:**

$$[\![\lambda x[P(x)]\ ]\!]^{M,g} = \{u : [\![P(x)]\!]^{M,g[u/x]} = 1\}$$

**But now we introduce worlds and times into the evaluation. And we discover that hardly anything formal changes:**

$$[\![\lambda x[P(x)]\ ]\!]^{M,w,i,g} = \{u : [\![P(x)]\!]^{M,w,i,g[u/x]} = 1\}$$

**it is now the property of being P, period. No mention of "actually". It is now "being P in all worlds", etc.**

**And this has a variety of consequences.**

**Consider**

**□ λx[P(x)] (m)] -- which is more properly written □λx[P(x)] (m)**

**vs.**

**□P(m)**

**In the first one, we find the value of *m* in the actual world (or whatever world we are using for evaluation). We say of that object that it has property P. And that this is necessary, and therefore holds in every world. So whoever *m* is in the actual world has P in every world.**

**The second one says that, for every world, whatever *m* designates in a world is P in that world.**

**Obviously, if *m* is a rigid designator, these amount to the same thing. But if *m* stands for 'the PM of Canada', and P stands for 'is leader of the Canadian Parliament', then the first one says "Paul Martin is leader of the Canadian Parliament" is true in every world, whereas the second one says "The PM of Canada is leader of the Canadian Parliament"**

**□P(m) would come from λx[□P(x)] (m)**

**A piece of terminology:**

**Montague introduced the convention that when you are talking about individual lexical terms that you don't have any special representations of, you simply add a prime to the word, and that means "the semantic representation of __, whatever it is". For example, for the English word *smiled* you designate the semantic representation as smiled´. (Actually, in Montague you would boldface the word and add the prime, instead of a squiggly underline. But these overheads are already boldfaced. Our book gives this convention on p.401, rule (24).)**