CHAPTER TWO: THE λ OPERATOR

2.1. Syntax of the λ-operator.

We will now discuss the λ-operator. The syntax of the λ-operator was given as follows:

> Functional abstraction:
> If $x \in VAR_a$ and $\beta \in EXP_b$ then $\lambda x\beta \in EXP_{<a,b>}$

If x is a variable of any type a and β an expression of any type b, then λxβ is an expression of the type of functions from a-entities into b-entities.

Some examples:

Let $x \in VAR_e$ and $P \in EXP_{<e,t>}$.
Then $(P(x)) \in EXP_t$.
Given that $x \in VAR_e$ and $(P(x)) \in EXP_t$, it follows that:
$\lambda x(P(x)) \in EXP_{<e,t>}$
Given that $(P(x)) \in EXP_t$, also $\neg(P(x)) \in EXP_t$, hence:
$\lambda x\neg(P(x)) \in EXP_{<e,t>}$

Let $Q \in EXP_{<e,t>}$. Then $(Q(x)) \in EXP_t$, and
$((P(x)) \wedge (Q(x))) \in EXP_t$. Then:
$\lambda x((P(x)) \wedge (Q(x))) \in EXP_{<e,t>}$

Let $P \in VAR_{<e,t>}$, $RONYA \in CON_e$, then $(P(RONYA)) \in EXP_t$. Hence:
$\lambda P(P(RONYA)) \in EXP_{<<e,t>,t>}$

Let $GIRL \in CON_{<e,t>}$, $P \in VAR_{<e,t>}$, $x \in VAR_e$.
Then $(GIRL(x)) \in EXP_t$, $(P(x)) \in EXP_t$, hence
$((GIRL(x)) \rightarrow (P(x))) \in EXP_t$.
Thus $\forall x((GIRL(x)) \rightarrow (P(x))) \in EXP_t$. Hence:
$\lambda P\forall x((GIRL(x)) \rightarrow (P(x))) \in EXP_{<<e,t>,t>}$

Let $Q \in VAR_{<e,t>}$. Then also $\forall x((Q(x)) \rightarrow (P(x))) \in EXP_t$.
Hence:
$\lambda P\forall x((Q(x)) \rightarrow (P(x))) \in EXP_{<<e,t>,t>}$
Now, we can apply the rule of forming λ-abstracts to this expression:
Since $Q \in VAR_{<e,t>}$ and $\lambda P\forall x((Q(x)) \rightarrow (P(x))) \in EXP_{<<e,t>,t>}$, we can abstract over variable Q:
$\lambda Q\lambda P\forall x((Q(x)) \rightarrow (P(x))) \in EXP_{<<e,t>,<<e,t>,t>>}$

Let $T,U \in VAR_{<<e,t>,t>}$ and $P \in VAR_{<e,t>}$
Then $(T(P)) \in EXP_t$ and $(U(P)) \in EXP_t$, hence
$((T(P)) \wedge (U(P))) \in EXP_t$.
Then:
$\lambda P((T(P)) \wedge (U(P))) \in EXP_{<<e,t>,t>}$

And:

$\lambda T\lambda P((T(P)) \wedge (U(P))) \in EXP_{<<<e,t>,t>,<<e,t>,t>>}$

And:

$\lambda U\lambda T\lambda P((T(P)) \wedge (U(P))) \in EXP_{<<<e,t>,t>,<<<e,t>,t>,<<e,t>,t>>>}$

Let $R,S \in VAR_{<e,<e,t>>}$, $x,y \in VAR_e$
Then $((R(y))(x)) \in EXP_t$ and $((S(y))(x)) \in EXP_t$.
Hence $(((R(y))(x)) \wedge ((S(y))(x)) \in EXP_t$.
Then:

$\lambda x(((R(y))(x)) \wedge ((S(y))(x)) \in EXP_{<e,t>}$.

And:

$\lambda y\lambda x(((R(y))(x)) \wedge ((S(y))(x)) \in EXP_{<e,<e,t>>}$.

Furthermore:

$\lambda R\lambda y\lambda x(((R(y))(x)) \wedge ((S(y))(x)) \in EXP_{<<e,<e,t>>,<e,<e,t>>>}$.

And:

$\lambda S\lambda R\lambda y\lambda x(((R(y))(x)) \wedge ((S(y))(x)) \in EXP_{<<e,<e,t>>,<<e,<e,t>>,<e,<e,t>>>>}$.

2.2. Semantics of the λ-operator.

The semantics of the λ-operator is given as follows:

> Functional abstraction.
> If $x \in VAR_a$ and $\beta \in EXP_b$ then:
> $[\![\lambda x\beta]\!]_{M,g} = h$
> where h is the unique function in $(D_a \rightarrow D_b)$ such that:
> for every $d \in D_a$: $h(d) = [\![\beta]\!]_{M,g^d_x}$
>
> That function from a-entities into b-entities that assigns to every $d \in D_a$ as value $[\![\beta]\!]_{M,g^d_x}$.

Let us look at some examples.b

⇒Let $x \in VAR_e$ and $P \in EXP_{<e,t>}$.

$\lambda x.P(x) \in EXP_{<e,t>}$

$[\![\lambda x.P(x)]\!]_{M,g} = h$,
where h is that function from D_e into D_t, i.e. in $(D \rightarrow \{0,1\})$ such that:
for every $d \in D$: $h(d) = [\![P(x)]\!]_{M,g^d_x}$

This means that h is that function in $(D \rightarrow \{0,1\})$ such that:
for every $d \in D$: $h(d) = [\![P]\!]_{M,g^d_x}([\![x]\!]_{M,g^d_x})$

i.e. h is that function in $(D \rightarrow \{0,1\})$ such that:
for every $d \in D$: $h(d) = F(P)(g^d_x(x))$

so h is that function in $(D \rightarrow \{0,1\})$ such that:
for every $d \in D$: $h(d) = F(P)(d)$

This means that for every $d \in D$: $h(d)=1$ iff $F(P)(d)=1$, and hence that $h = F(P)$.

Thus:
$[\![\lambda x.P(x)]\!]_{M,g} = F(P) = [\![P]\!]_{M,g}$

It is useful to read variable x as (generic) 'you'. Then we read the expression $\lambda x.P(x)$ as:
the property that you have iff you have P

We see that the semantics of the λ-operator tells us, as it should, that that property is P.

$\Rightarrow$Let $x \in VAR_e$ and $P \in EXP_{<e,t>}$.
$\lambda x.\neg P(x) \in EXP_{<e,t>}$

$[\![\lambda x.\neg P(x)]\!]_{M,g} = h$,
where h is that function in $(D \rightarrow \{0,1\})$ such that:
for every $d \in D$: $h(d) = [\![\neg P(x)]\!]_{M,g^d_x}$

i.e. h is the function in $(D \rightarrow \{0,1\})$ such that:
for every $d \in D$: $h(d) = \neg([\![P(x)]\!]_{M,g^d_x})$

Thus h is the function in $(D \rightarrow \{0,1\})$ such that:
for every $d \in D$: $h(d) = \neg([\![P]\!]_{M,g^d_x}([\![x]\!]_{M,g^d_x}))$

i.e. h is the function in $(D \rightarrow \{0,1\})$ such that:
for every $d \in D$: $h(d) = \neg(F(P)(g^d_x(d)))$

Hence: h is the function in $(D \rightarrow \{0,1\})$ such that:
for every $d \in D$: $h(d) = \neg(F(P)(d))$

Now $\neg = \{<0,1>,<1,0>\}$
Hence h is that function in $(D \rightarrow \{0,1\})$ such that:
for every $d \in D$: $h(d)=1$ iff $F(P)(d)=0$

This means that $[\![\lambda x.\neg P(x)]\!]_{M,g}$ is the property that you have iff you don't have P.

Thus, if we use constant $WALK \in CON_{<e,t>}$ as the representation for *walk*, then $\lambda x.\neg WALK(x)$ is an ideal representation for *not walk*.

⇒Let P,Q ∈ $EXP_{<e,t>}$, x ∈ VAR_e.
λx.P(x) ∧ Q(x) ∈ $EXP_{<e,t>}$

$[\![\lambda x.P(x) \wedge Q(x)]\!]_{M,g}$ = h,
where h is that function in (D → {0,1}) such that:
for every d ∈ D: h(d) = $[\![P(x) \wedge Q(x)]\!]_{M,g^d_x}$

$[\![P(x) \wedge Q(x)]\!]_{M,g^d_x} = \wedge(<[\![P(x)]\!]_{M,g^d_x},[\![Q(x)]\!]_{M,g^d_x}> =$
$\wedge(<[\![P]\!]_{M,g^d_x}([\![x]\!]_{M,g^d_x}),[\![Q]\!]_{M,g^d_x}([\![x]\!]_{M,g^d_x})> =$
∧(<F(P)(d),F(Q)(d)>)

Hence h is that function in (D → {0,1}) such that:
for every d ∈ D: h(d) = ∧(<F(P)(d),F(Q)(d)>)

Thus, h is that function in (D → {0,1}) such that:
for every d ∈ D: h(d)=1 iff F(P)(d)=1 and F(Q)(d)=1

As we have seen, this function is the characteristic function of the set {d ∈ D: F(P)(d)=1 and F(Q)(d)=1} = {d ∈ D: d ∈ F(P) and d e F(Q)} = F(P) ∩ F(Q)

$[\![\lambda x.P(x) \wedge Q(x)]\!]_{M,g}$ is the property that you have iff you have both property P and property Q.

Thus, if we choose constants WALK, TALK ∈ $CON_{<e,t>}$ as representations for *walk* and *talk*, λx.WALK(x) ∧ TALK(x) represents *walk and talk*.

⇒Let P ∈ $VAR_{<e,t>}$, RONYA ∈ CON_e.
λP.P(RONYA) ∈ $EXP_{<<e,t>,t>}$

$[\![\lambda P.P(RONYA)]\!]_{M,g}$ = that function h:((D → {0,1}) → {0,1}) such that:
for every K ∈ (D → {0,1}): h(K) = $[\![P(RONYA)]\!]_{M,g^K_P}$ (note, again, that $g^K_P(P) = K$)

= that function h such that:
for every K ∈ $D_{<e,t>}$: h(K) = $[\![P]\!]_{M,g^K_P}([\![RONYA]\!]_{M,g^K_P})$

= that function h such that:
for every K ∈ $D_{<e,t>}$: h(K) = $g^K_P(P)(F(RONYA))$

= that function h such that:
for every K ∈ $D_{<e,t>}$: h(K) = K(F(RONYA))

= that function h such that:
for every K ∈ $D_{<e,t>}$: h(K)=1 iff K(FRONYA))=1

K is the characteristic function of a set of individuals, set theoretically h is that function such that: for every K ∈ $D_{<e,t>}$: h(K)=1 iff F(RONYA) ∈ K

h itself is the characteristic function of a set of properties (sets), namely:
$\{K \in D_{<e,t>}: h(K)=1\}$
Hence, h characterizes the set: {K: F(RONYA) ∈ K}: the set of all sets that contain F(RONYA), or: the set of all properties that F(RONYA) has.

Thus $[\![\lambda P.P(RONYA)]\!]_{M,g}$ is the set of all properties that Ronya has.

⇒Let GIRL ∈ $CON_{<e,t>}$, P ∈ $VAR_{<e,t>}$, x ∈ VAR_e.
$\lambda P.\forall x[GIRL(x) \rightarrow P(x)] \in EXP_{<<e,t>,t>}$

$[\![\lambda P.\forall x[GIRL(x) \rightarrow P(x)]]\!]_{M,g}$ = h: $D_{<e,t>} \rightarrow \{0,1\}$, where
h is that function from properties into truth values such that:
for every K ∈ $D_{<e,t>}$: $h(K) = [\![\forall x[GIRL(x) \rightarrow P(x)]]\!]_{M,g^K_P}$

$[\![\forall x[GIRL(x) \rightarrow P(x)]]\!]_{M,g^K_P} = 1$ iff
for every d ∈ D: $[\![GIRL(x) \rightarrow P(x)]\!]_{M,g^{K\,d}_{P\,x}} = 1$
This is evaluated relative to the assignment function which is the the result of resetting the value of P to K and of x to d in g: $g^{K\,d}_{P\,x}$

iff for every d ∈ D: F(GIRL)(d)=0 or K(d)=1

Hence, h is that function such that: for every K ∈ $D_{<e,t>}$:
h(K)=1 iff for every d ∈ D: F(GIRL)(d)=0 or K(d)=1

Again, using the set theoretic equivalence:
for every K: h(K)=1 iff for every d ∈ F(GIRL): d ∈ K
i.e. for every K: h(K)=1 iff F(GIRL) ⊆ K.

Thus, h characterizes the set:
$\{K \in D_{<e,t>}: F(GIRL) \subseteq K\}$: the set of all sets that F(GIRL) is a subset of. This is
{K: for every d ∈ F(GIRL): d ∈ K}, the set of all properties that every girl has.

Hence $[\![\lambda x.\forall x[GIRL(x) \rightarrow P(x)]]\!]_{M,g}$ is the set of all properties that every girl has.

⇒Let P,Q ∈ $VAR_{<e,t>}$, x ∈ VAR_e.
$\lambda Q\lambda P.\forall x[Q(x) \rightarrow P(x)] \in EXP_{<<e,t>,<<e,t>,t>>}$

$[\![\lambda Q\lambda P.\forall x[Q(x) \rightarrow P(x)]]\!]_{M,g}$ = h,
where h is that function h: ($D_{<e,t>} \rightarrow D_{<<e,t>,t>}$) such that:
For every L ∈ $D_{<e,t>}$: $h(L) = [\![\lambda P.\forall x[Q(x) \rightarrow P(x)]]\!]_{M,g^L_Q}$ b

It is easy to see that:
$[\![\lambda P.\forall x[Q(x) \rightarrow P(x)]]\!]_{M,g^L_Q}$ = that function j such that:
for every K: j(K)=1 iff L ⊆ K

Hence, h is that function such that for every $L \in D_{<e,t>}$, for every $K \in D_{<e,t>}$:
$(h(L))(K)=1$ iff $L \subseteq K$

h characterizes a set of ordered pairs: $\{<K,L>: (h(L))(K)=1\}$, hence h characterizes the set: $\{<K,L>: L \subseteq K\}$. This is the subset relation, the relation that holds between two sets K and L iff $L \subseteq K$.

Thus $[\![\lambda Q \lambda P.\forall x[Q(x) \rightarrow P(x)]]\!]^{b}_{M,g}$ is the relation that holds between two sets P and Q iff Q is a subset of P (i.e. iff every Q is a P).

$\Rightarrow$ $\lambda Q \lambda P.\exists x[Q(x) \wedge P(x)] \in EXP_{<<e,t>,<<e,t>,t>>}$

$[\![\lambda Q \lambda P.\exists x[Q(x) \wedge P(x)]]\!]^{b}_{M,g} = h$,
where h is that function h: $(D_{<e,t>} \rightarrow D_{<<e,t>,t>})$ such that:
For every $L \in D_{<e,t>}$: $h(L) = [\![\lambda P.\exists x[Q(x) \wedge P(x)]]\!]_{M,g^{L}_{Q}}$

$[\![\lambda P.\exists x[Q(x) \wedge P(x)]]\!]_{M,g^{L}_{Q}}$ = that function j such that:
for every K: $j(K)=1$ iff for some $d \in D$: $L(d)=1$ and $K(d)=1$

Hence, h is that function such that for every $L \in D_{<e,t>}$, for every $K \in D_{<e,t>}$:
$(h(L))(K)=1$ iff $L \cap K \neq \emptyset$

h characterizes $\{<K,L>: L \cap K \neq \emptyset\}$, the relation that holds between two sets K and L iff $L \cap K \neq \emptyset$

Thus $[\![\lambda Q \lambda P.\exists x[Q(x) \wedge P(x)]]\!]_{M,g}$ is the relation that holds between two sets P and Q iff the intersection of Q and P is not empty (i.e. iff some Q is a P).

$\Rightarrow$ Let $A,B \in CON_{<<e,t>,t>}$ and $P \in VAR_{<e,t>}$
$\lambda P.A(P) \wedge B(P) \in EXP_{<<e,t>,t>}$

$[\![\lambda P.A(P) \wedge B(P)]\!]_{M,g} = h: D_{<e,t>} \rightarrow \{0,1\}$, where
h is that function such that:
for every $K \in D_{<e,t>}$: $h(K) = [\![A(P) \wedge B(P)]\!]_{M,g^{K}_{P}}$

i.e. h is that function such that:
for every $K \in D_{<e,t>}$: $h(K)=1$ iff $K \in F(A)$ and $K \in F(B)$

This means that h characterizes the set:
$\{K: K \in F(A)$ and $K \in F(B)\}$, hence h characterizes $F(A) \cap F(B)$

$[\![\lambda P.A(P) \wedge B(P)]\!]_{M,g}$ is the set of all properties in $A \cap B$

⇒ Let U,T ∈ $VAR_{<<e,t>,t>}$ and P ∈ $VAR_{<e,t>}$.
λUλTλP.T(P) ∧ U(P) ∈ $EXP_{<<<e,t>,t>,<<<e,t>,t>,<<e,t>,t>>>}$

$[\![\lambda U\lambda T\lambda P.T(P) \wedge U(P)]\!]_{M,g}$ = h,
where for every B ∈ $D_{<<e,t>,t>}$: $h(B) = [\![\lambda T\lambda P.T(P) \wedge U(P)]\!]_{M,g^B_U}$

$[\![\lambda T\lambda P.T(P) \wedge U(P)]\!]_{M,g^B_U}$ = j,
where for every A ∈ $D_{<<e,t>,t>}$: $j(A) = A \cap B$

Thus $[\![\lambda U\lambda T\lambda P.T(P) \wedge U(P)]\!]_{M,g}$ = h,
where for every B, A ∈ $D_{<<e,t>,t>}$: $(h(B))(A) = A \cap B$

Hence, using the set theoretic correspondence,

$[\![\lambda U\lambda T\lambda P.T(P) \wedge U(P)]\!]_{M,g}$ = the function which maps any two sets of sets A and B onto their intersection.

Thus, if λP.P(RONYA) is the representation of the noun phrase *ronya*, interpreted as {K: F(RONYA) ∈ K}, the set of all properties that Ronya has, and λP.P(PIM) is the representation of the noun phrase *Pim*, interpreted as {K:F(PIM) ∈ K}, the set of all properties that Pim has, we can represent noun phrase conjunction as:
λUλTλP.T(P) ∧ U(P). This will give a representation for *Ronya and Pim* which is interpreted as {K: F(RONYA) ∈ K and F(PIM) ∈ K}, the set of all properties that both Ronya and Pim have.

⇒ Let X,Y ∈ $CON_{<e,<e,t>>}$, x,y ∈ VAR_e
λyλx.X(x,y) ∧ Y(x,y) ∈ $EXP_{<e,<e,t>>}$.

$[\![\lambda y\lambda x.X(x,y) \wedge Y(x,y)]\!]_{M,g}$ = h: $D \rightarrow D_{<e,t>}$,
where for every b ∈ D: $h(b) = [\![\lambda x.X(x,y) \wedge Y(x,y)]\!]_{M,g^b_y}$

$[\![\lambda x.X(x,y) \wedge Y(x,y)]\!]_{M,g^b_y}$ = j ∈ $D_{<e,t>}$,
where for every a ∈ D: j(a)=1 iff <a,b> ∈ F(X) and <a,b> ∈ F(Y)

Hence h is that function such that:
for every b ∈ D for every a ∈ D:
h(b)(a)=1 iff <a,b> ∈ F(X) and <a,b> ∈ F(Y)

This means that h characterizes the set $F(X) \cap F(Y)$

Hence $[\![\lambda y\lambda x.X(x,y) \wedge Y(x,y)]\!]_{M,g} = F(X) \cap F(Y)$

⇒Let R,S ∈ $VAR_{<e,<e,t>>}$, x,y e VAR_e.
λSλRλyλx.R(x,y) ∧ S(x,y) ∈ $EXP_{<<e,<e,t>>,<<e,<e,t>>,<e,<e,t>>>>}$.

$[\![\lambda S\lambda R\lambda y\lambda x.R(x,y) \wedge S(x,y)]\!]_{M,g}$ is that function h such that:
for every B,A ∈ $D_{<e,<e,t>>}$: (h(B))(A) = A ∩ B

Thus, if KISSED is the representation of *kissed* and HUGGED the representation of *hugged*, we can represent transitive verb phrase conjunction *and* as λSλRλyλx.R(x,y) ∧ S(x,y); its interpretation will take the set of pairs that stand in the kiss relation and the set of pairs that stand in the hug relation, and map them onto the set of pairs that stand both in the kiss and the hug relation.

⇒ Let P ∈ $VAR_{<e,t>}$, x ∈ VAR_e and OLD ∈ $CON_{<e,t>}$
λPλx.P(x) ∧ OLD(x) ∈ $EXP_{<<e,t>,<e,t>>}$

$[\![\lambda P\lambda x.P(x) \wedge OLD(x)]\!]_{M,g}$ = h:$(D_e \rightarrow D_t) \rightarrow (D_e \rightarrow D_t)$
where for every K ∈ $D_{<e,t>}$ and for every d ∈ D:
h(K)(d)=1 iff K(d)=1 and F(OLD)(d)=1

i.e. $[\![\lambda P\lambda x.P(x) \wedge OLD(x)]\!]_{M,g}$ = h such that:
for every K ∈ $D_{<e,t>}$ and for every d ∈ D:
h(K)(d)=1 iff d ∈ K and d ∈ F(OLD)

In other words:

$[\![\lambda P\lambda x.P(x) \wedge OLD(x)]\!]_{M,g}$ = h such that for every K ∈ $D_{<e,t>}$:
h(K) = K ∩ F(OLD)

[Note, we are not here dealing with the comparison set dependency of *old*, we treat it, for ease, as an expression whose comparision is *fixed*. This is for ease of exposition only.]

Of course, a lot of these expressions (or rather their interpretations) are familiar from the discussion above of sentence (1):

(1) Some old man and every girl kissed and hugged Ronya.

We gave this the following representation:

[SOME(OLD(MAN) AND_1 EVERY(GIRL)] ([KISSED AND_2 HUGGED](RONYA))

This involves choosing the following translation of the lexical items into functional type logic:

girl	→	GIRL	$\in CON_{<e,t>}$
man	→	MAN	$\in CON_{<e,t>}$
ronya	→	RONYA	$\in CON_{e}$
kissed	→	KISSED	$\in CON_{<e,<e,t>>}$
hugged	→	HUGGED	$\in CON_{<e,<e,t>>}$
old	→	OLD	$\in CON_{<<e,t>,<e,t>>}$
every	→	EVERY	$\in CON_{<<e,t>,<<e,t>,t>>}$
some	→	SOME	$\in CON_{<<e,t>,<<e,t>,t>>}$
and$_1$	→	AND$_1$	$\in CON_{<<<e,t>,t>,<<<e,t>,t>,<<e,t>,t>>>}$
and$_2$	→	AND$_2$	$\in CON_{<<e,<e,t>>,<<e,<e,t>>,<e,<e,t>>>>}$

With the λ-operator, we do not use these constants as the translations of the corresponding lexical items, but we translate them as complex expressions that denote the functions the meaning postulates tell us they should denote (this means, of course, that the meaning postulates become irrelevant):

old	→	$\lambda P \lambda x.P(x) \wedge OLD(x)$	$\in EXP_{<<e,t>,<e,t>>}$
every	→	$\lambda Q \lambda P.\forall x[Q(x) \rightarrow P(x)]$	$\in EXP_{<<e,t>,<<e,t>,t>>}$
some	→	$\lambda Q \lambda P.\exists x[Q(x) \wedge P(x)]$	$\in EXP_{<<e,t>,<<e,t>,t>>}$
and$_1$	→	$\lambda U \lambda T \lambda P.T(P) \wedge U(P)$	$\in EXP_{<<<e,t>,t>,<<<e,t>,t>,<<e,t>,t>>>}$
and$_2$	→	$\lambda S \lambda R \lambda y \lambda x.R(x,y) \wedge S(x,y)$	$\in EXP_{<<e,<e,t>>,<<e,<e,t>>,<e,<e,t>>>>}$

The advantage of these expressions is that we can read the meaning off the type logical expression:

$\lambda P \lambda x.P(x) \wedge OLD(x)$ is the function which takes any property P and maps it onto the property $\lambda x.P(x) \wedge OLD(x)$, the property that you have if you have P and you are old.

$\lambda Q \lambda P.\forall x[Q(x) \rightarrow P(x)]$ is the relation that holds between sets P and Q if every Q is a P.

$\lambda Q \lambda P.\exists x[Q(x) \wedge P(x)]$ is the relation that holds between sets P and Q if some Q is a P.

$\lambda U \lambda T \lambda P.T(P) \wedge U(P)$ is the function which takes any two sets of properties A and P and maps them onto $\lambda P.U(P) \wedge T(P)$, the set of properties that are in U and in T.

$\lambda S \lambda R \lambda y \lambda x.R(x,y) \wedge S(x,y)$ is the function which takes two relations R and S and maps them onto the relation $\lambda y \lambda x.R(x,y) \wedge S(x,y)$, the relation that x and y stand in iff they stand both in relation R and relation S.

Thus we have gained the advantage of making the representations of the lexical items more perspicuous. One can't really say that, as such, we have gained a lot of advantage in making the representation of (1) more perspicuous, it now becomes (in infix notation):

[[λQλP.∃x[Q(x) ∧ P(x)](λPλx.P(x) ∧ OLD(x)(MAN))]
[λUλTλP.T(P) ∧ U(P)] [λQλP.∀x[Q(x) → P(x)](GIRL)]]
([KISSED [λSλRλyλx.R(x,y) ∧ S(x,y)] HUGGED](RONYA))

Or, undoing infix notation:

[[λUλTλP.T(P) ∧ U(P))](λQλP.∀x[Q(x) → P(x)](GIRL))
(λQλP.∃x[Q(x) ∧ P(x)](λPλx.P(x) ∧ OLD(x)(MAN)))]
([λSλRλyλx.R(x,y) ∧ S(x,y)](HUGGED)(KISSED)(RONYA))

This, in fact, seems even worse than:

[AND_1(EVERY(GIRL))(SOME(OLD(MAN)))](AND$_2$(HUGGED)(KISSED)(RONYA))

Yet, there is a big advantage, and that is, that we can use the logical properties of λ-abstraction and functional application to find in a simple way a more readable expression of functional type logic which is logically equivalent to this expression, i.e. we can **reduce** the representation, by using some rules concerning λ-abstraction. This we will do in section 2.5.

2.3. Working with lambdas and types

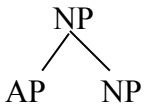
We have illustrated the syntax and semantics of the λ-operator with λ-expressions whose interpretation we claim are a proper for deriving the truth conditions of the sentence we started out with. But we haven't discussed how you, that is *you*, actually get to these interpretations yourself. We will systematically investigate this issue in chapter Three by discussing backward λ-conversion. But we will here already look at how thinking about λ-expressions and types actually guide you towards the right interpretation in simpler cases.

⇒ We start with *old*.
We already discussed our strategy for providing a semantically interpreted grammar for attributival adjectives:
-syntactically they are NP-modifiers, that combine with an NP and yield a complex NP.
-Semantically they take the interpretation of the NP they combine with of type <e,t> and yield an <e,t> interpretation of the complex NP they are part of.
-So we assume that the interpretation of *old* is in the domain of type <<e,t>,<e,t>>, a function that maps a set onto a set.
And now we are looking for an expression α of type <<e,t>,<e,t>> to represent this interpretation.

This expression α will give us the interpretation for the complex expression *old man* via functional application:

$\alpha \in EXP_{<<e,t>,<e,t>>}$ $MAN \in EXP_{<e,t>}$
hence: $(\alpha(MAN)) \in EXP_{<e,t>}$

Let's be explicit about this. This means that we assume that in a structure:



The interpretation of the AP is an expression of type <<e,t>,<e,t>> that combines with the interpretation of type <e,t> of the lower NP *via functional application*, yielding an interpretation of type <e,t> for the complex constituent. Thus, the semantic interpretation of the tree itself is functional application.

We come to the interpretation of *old*. We will ignore subtleties of the theory of adjectives here and assume that in *old man*, *old* is an intersective adjective, and this means that that we take (2a) and (2b) to be equivalent:

(2) a. Fred is an old man.
b. Fred is a man and Fred is old.

In (2b) we see the predicative adjective which we assume has an interpretation as a one place predicate OLD of type <e,t>. This means that we can write the truth conditions of (2b) in type logic as (2c): (I am ignoring irrelevant brackets):

(2) c. MAN(FRED) ∧ OLD(FRED)

where OLD ∈ $CON_{<e,t>}$.

We will use this same constant in providing an interpretation for the attributival adjective at type <<e,t>,<e,t>>.

Finding that interpretation is a question of two things:
1. Realizing what the interpretation of the attributive adjective *does*.
2. Working in its intersective meaning.

We start with the first. The expression we are after needs to apply to a set of individuals of type <e,t>, and yield a set of individuals of type <e,t>.
In a functional expression we typically have
-one part that describes the input of the function,
this part is typically a sequence of lambda operators
-and one part which is the description of the function
this part is typically an expression containing variables bound by those lambdas.

We separate the two part by putting a dot in between them (although exactly where we put the dot is rather variable, which is ok, since it is a harmless convention).

So, we are concerned with a function of type <<e,t>,<e,t>>.
We represent the input requirement by chosing a variable P ∈ $VAR_{<e,t>}$ and starting the expression we are after with λP.

λP.

The output is a function of type <e,t>, the nature of which is still to be determined, that function will take individuals onto truth values. To describe this function we chose a second variable $x \in VAR_e$ and continue the expression with λx:

λPλx.

For the types to fit, what must follow the dot is an expression φ of type t.

λP.λxφ

Why?
Because then λx.φ is an expression of type <e,t>, and indeed λP,λxφ is an expression of type <<e,t>,<e,t>>. Thinking about the the semantics as being from <e,t> to <e,t>, we really want to think of this as λP.λxφ, you take a predicate of type <e,t> as input, indicated by λP, and you get a predicate of type <e,t> λx.φ as output.

Now this expression λP.λxφ tells us which function of type <<e,t>,<e,t>> we want to define. Since we are interested in the meaning of the attributive adjective *old*, the description of the function should be formulated in terms of variables P and x and constant OLD: So, φ = ...P...x...OLD...

λP.λx ...P...x...OLD...

The model of it, is, of course given by (2c): ((MAN(FRED)) ∧ (OLD(FRED)))
The function λP.λx ...P...x...OLD... should apply to MAN, to give the interpretation of *old man*, and this, in its turn will apply to FRED, to get the meaning of *Fred is an old man*. And when you apply this function λP.λx ...P...x...OLD... to the predicate $MAN \in CON_{<e,t>}$ and to individual $FRED \in CON_e$, you should get as meaning (2c):

(2) c. MAN(FRED) ∧ OLD(FRED)

This means that, if we replace in (2c) FRED by x and MAN by P, we have the perfect description of the function we are after:

P(x) ∧ OLD(x)

This is the description we add this after λPλx:

λP.λxP(x) ∧ OLD(x)

We check that this is of the right type:
$x \in VAR_e$ and P(x) ∧ OLD(x) $\in EXP_t$, hence λx.P(x) ∧ OLD(x) $\in EXP_{<e,t>}$
$P \in VAR_{<e,t>}$ and λx.P(x) ∧OLD(x) $\in EXP_{<e,t>}$ hence λPλx.P(x) ∧OLD(x) $\in EXP_{<<e,t>,<e,t>>}$

Reading the expression:

P(x) ∧ OLD(x)
you have *that* property and *you* are old

λx.P(x) ∧ OLD(x)
the property that you have if you have *that* property and you are old =
the set of individuals that have *that* property and are old

λPλx.P(x) ∧ OLD(x)
the function that maps any property onto the set of individuals that have that property and are old

We now have derived the following interpretations:

old → λPλx.P(x) ∧ OLD(x) P ∈ $VAR_{<e,t>}$, x ∈ VAR_e, OLD ∈ $CON_{<e,t>}$
man → MAN MAN ∈ $CON_{<e,t>}$

And with the assumption that the adjunction tree is interpreted as functional appliciation, we have also derived an interpretation for the complex NP:

old man → (λPλx.P(x) ∧ OLD(x)(MAN)) ∈ $EXP_{<e,t>}$

We wouldn't mind being able to write this in a simpler way. This is what λ-conversion will do for us later.

⇒ Next we are concerned with determiners *every* and *some*.
The type of these we already fixed in Foundations, but now written in curried form: <<e,t>,<<e,t>,t>> of 2-place relations between sets of individuals.

But grammatically, we think of generalized quantifiers functionally: *every* and *some* need to combine with a noun interpretation of type <e,t>, like CAT, and a verbal interpret of type <e,t>, like PURR, to derive an interpretation of type t. Of course, as far as the semantics goes, we can take the standard semantics as our basis:

(3) a. Some cat purrs ∃x[CAT(x) ∧ PURR(x)]
b. Every cat purrs ∀x[CAT(x) → PURR(x)]

So we need an interpretation that takes an <e,t> interpretation
- we write λQ with Q ∈ $VAR_{<e,t>}$ - and then takes another <e,t> interpretation
- we write λP with P ∈ $VAR_{<e,t>}$ - and gives a description of type t:

λQλP. ....Q...P...

Note here that λQ is going to be the argument of the relation associated with the noun that the determiner combines withm the first argument in, while λP is going to be assicated with the verbal predicate. Mnemonically the earlier letter in the alphabet corresponds with the last argument in.

We read the descriptions off the semantics given in (3), and we associate Q with CAT and P with PURR, so the description of the function becomes:

∃x[Q(x) ∧ P(x) and ∀x[Q(x) → P(x)]

Putting functional input and function description together gives us:

some → λQλP.∃x[Q(x) ∧ P(x)]
every → λQλP.∀x[Q(x) → P(x)]

For grammatical reasons we read them functionally:
some is a function that applies to CAT, and then applies to PURR to give you truthvalue 1 iff some cat purrs.
every is a function that applies to CAT, and then applies to PURR to give you truthvalue 1 if every cat purrs.

Note that we could just as well have written this in Generalized Quantifier notation, with the interpretations given as in Foundations.

some → λQλP.SOME[Q,P]
every → λQλP.EVERY[Q,P]

These denote exactly the same relations between sets of sets, so the Frege-Tarski style quantifier notation has nothing to do with it.

Can we write:
every → λQλP.Q ⊆ P]
some → λQλP. (Q ∩ P) = Ø

We can't, but only because we don't have ⊆, ∩ and Ø in our language TL.
And there is no reason *not* to add these to the language. The simplest way to to that is by definition:

We use := for *is by definion* and define:

⊆ := λQλP.∀x[Q(x) → P(x)]
∩ := λQλPλx.Q(x) ∧ P(x)
Ø := λx.¬(x = x)

Now the above expressions using the set theoretic operations are defined in the type theory (for type <e,t>).

We check that we have the right types:

$P \in VAR_{<e,t>}$ and $\exists x[Q(x) \wedge P(x)] \in EXP_t$ so $\lambda P.\exists x[Q(x) \wedge P(x)] \in EXP_{<<e,t>,t>}$
$Q \in VAR_{<e,t>}$ and $\lambda P.\exists x[Q(x) \wedge P(x)] \in EXP_{<<e,t>,t>}$
so $\lambda Q\lambda P.\exists x[Q(x) \wedge P(x)] \in EXP_{<<e,t>,<<e,t>,t>>}$

With this we have derived interpretations for the DPs:

some old man → $(\lambda Q\lambda P.\exists x[Q(x) \wedge P(x)]((\lambda P\lambda x.P(x) \wedge OLD(x)(MAN)))) \in EXP_{<<e,t>,t>}$
every girl → $(\lambda Q\lambda P.\forall x[Q(x) \rightarrow P(x)](GIRL) \in EXP_{<<e,t>,t>}$

The fact that this looks complex is not important, because it is only because I am still refusing to use λ-conversion to reduce these expressions to simpler expressions. The important thing is that we succesfully have provided interpretations for the complex DPs, and, with an argument by intimidation, interpretations that derive the right meaning for the sentence, as we will see.

⇒ Next we need to be concerned with conjunction.
I will do this more systematically in the next chapter. But the thinking is the same:
Let's think about predicate conjunction: the interpretation of *walk and purr*.
WALK ∈ $CON_{<e,t>}$ and PURR ∈ $CON_{<e,t>>}$ and we know the truthc conditions of (3a):

(3) a.Ronya walks and purrs.
b. WALK(RONYA) ∧ PURR(RONYA)

and takes two one place predicates of type <e,t> and yields a one place predicate of type <e,t>:

λQλPλx.φ

The description φ is derived from (3b) in the same way as we already knew:

Q(x) ∧ P(x)

Combining the two gives:

λQλPλx.Q(x) ∧ P(x)

or, with the new definition of ∩:

λQλP.Q ∩ P

The function that takes two sets and maps them onto their intersection.

Can we do the same for relations, as in *kissed and hugged*?
Sure! Here conjunction should take two relations of type <e,<e,t>> and maps them onto a relation of type <e,<e,t>>. We take (4) as our model:

(4) a. Pim kissed and hugged Ronya
b. KISSED(PIM, RONYA) ∧ HUGGED(PIM, RONYA)

We take two relational variables R and S and two individual variable x and y, and we know that *and* should do the following:

λSλR.λyλxφ where λyλx.φ is an expression of type <e,<e,t>>

Again, we associate KISSED with R and HUGGED with S and PIM with x and RONYA with y and get:

and$_{<<e,<e,t>>, <<e,<e,t>>, <e, <e,t>>>>}$ $\lambda S\lambda R.\lambda y\lambda x R(x, y) \wedge S(x,y)$

The function that maps any two relations R and S onto the relation that holds between any x and y iff x kissed y and x hugged y.

$R(x, y) \wedge S(x,y)$	is of type t
$\lambda x R(x, y) \wedge S(x,y)$	is of type <e,t>
$\lambda y\lambda x R(x, y) \wedge S(x,y)$	is of type <e,<e,t>>
$\lambda R.\lambda y\lambda x R(x, y) \wedge S(x,y)$	is of type <<e,<e,t>>,<e,<e,t>>>
$\lambda S\lambda R.\lambda y\lambda x R(x, y) \wedge S(x,y)$	is of type <<e,<e,t>>, <<e,<e,t>>,<e,<e,t>>>>

⇒ We do the same with DP conjunction, but arguing the case intuitively is much easier to do after we have done λ-conversion.
But here we can easily argue by analogy.

We found for conjunction at type <<**e**,t>,<<**e**,t>,<**e**,t>>>: $\lambda Q\lambda P\lambda x.P(x) \wedge Q(x)$
We are now concerned with conjunction at type
<<**<e,t>**t>, <<**<e,t>**t> , <<**<e,t>**t>>. We see that we have <e,t> everywhere where we had e before. But that suggests that we can use *the very same expression*, as long as we interpret $x \in VAR_{<e,t>}$ and $P,Q \in VAR_{<<e,t>,t>}$.
And that is exactly correct. Instead, for mnemonic reasons we don't use x but $P \in VAR_{<e,t>}$, and instead of P and Q we use T and $U \in VAR_{<<e,t>,t>}$ (with T for *term*) and get:

and$_{<<<e,t>t>, <<<e,t>t> , <<e,t>t>>}$ $\lambda U\lambda T.\lambda P.T(P) \wedge U(P)$

2.4. Alphabetic variants

We specified the syntax and semantics of the λ-operator in the previous section:

> If $x \in VAR_a$ and $\beta \in EXP_b$, then $\lambda x\beta \in EXP_{<a,b>}$
> $[\![\lambda x\beta]\!]_{M,g}$ is the function $h \in (D_{M,a} \rightarrow D_{M,b})$ such that for all $d \in D_{M,a}$: $h(d) = [\![\beta]\!]_{M,g^d_x}$

We see that the λ-operator is a variable binding operation, and in this respect similar to the quantifiers. This means that notions of free versus bound variables and alphabetic variants familiar from quantification apply in the same way to the λ-operator:

-In $\lambda x\beta$, the outside occurrence of λx binds all occurrences of variable x that are free in β.
-Let $\beta[y/x]$ be the result of replacing all free occurrences of x in β by y.
Then $\lambda x\beta$ and $\lambda y\beta[y/x]$ are equivalent if the free occurrences of x in β and the free occurrences of y in $\beta[y/x]$ are identical, except for the label x or y.

In practice we take alphabetic variants a lot, simply to make what we are doing more readable.
Thus we may have an expression:

λPλx((P(x)) ∧ (SMART(y))

which we will write for clarity as:

λPλx.P(x) ∧ SMART(x)

And this we apply to an expression that also has P's and x's, say, λx.P(x).
Then we get an expression:

[λPλx.P(x) ∧ SMART(x)](λx.P(x))

As for quantifiers, the binder relations are as indicated, by the colours:

[λPλx.P(x) ∧ SMART(x)](λx.P(x))

What we see is that the yellow λx and x have nothing to do with the green λx and x, and the blue P is free.

As will see, when we come to λ-conversion, if we don't do anything, we will have to deal with formulas like:

λx.[λx.CAT(x)](x) ∧ SMART(x)

And that is not a problem, as long as we realize that this is:

λx.[λx.CAT(x)](x) ∧ SMART(x)

But, of course, it is very easy to make mistakes here.
For that reason we will almost automatically take alphabetic variants where necessary.

So, instead of using the expression:

[λPλx.P(x) ∧ SMART(x)](λx.CAT(x))

we take an alphabetic variant:

[λPλx.P(x) ∧ SMART(x)](λy.CAT(y))

(it doesn't matter which)

and this means that we only have to deal with:

λx.[λy.CAT(y)](x) ∧ SMART(x)

without variable collision.

And we do that systematically:

$\lambda\cup\lambda T\lambda P.T(P) \wedge U(P) + \lambda P.\forall x[CAT(x) \rightarrow P(x)]$?

take an alphabetic variant:

$\lambda\cup\lambda T\lambda P.T(P) \wedge U(P) + \lambda \mathbf{Q}.\forall x[CAT(x) \rightarrow \mathbf{Q}(x)]$

and then apply.

From Foundations:
Let α be an expression, x and y variables.
Let q_x be an occurrence of ∀x or ∃x or λx in α.
Let $\{ v_{1,x}, \ldots, v_{n,x}\}$ be the set of **all** occurrences of variable x bound by q_x in α.
(So q and v_1,...,v_n stand for nodes in the construction tree.)

We call $< q_x, v_{1,x}, \ldots, v_{n,x} >$ a **binding relation** in α.

Crucially, this means that, if $< q_x, v_{1,x}, v_{2,x} >$ is a binding relation in α, then $< q_x, v_{1,x} >$ is not a binding relation in α, it got to be **all and only** the bound occurrences to be called a binding relation.

Now take the construction tree for α, and binding relation $< q_x, v_{1,x}, \ldots, v_{n,x} >$ in α and:

1. replace $< q_x, v_{1,x}, \ldots, v_{n,x} >$ by $< q_y, v_{1,y}, \ldots, v_{n,y} >$.
2. adjust the nodes above in the tree accordingly.

This gives an expression which we can call: $\alpha_{<q_y, v_{1,y}, \ldots, v_{n,y}>}$

We define:

Expressions α and β are **basic alphabetic variants** iff
there are variables x and y and there is a binding relation $< q_x, v_{1,x}, \ldots, v_{n,x} >$ in α
such that 1. $\beta = \alpha_{<q_y, v_{1,y}, \ldots v_{n,y}>}$ and
2. $< q_y, v_{1,y}, \ldots, v_{n,y} >$ is a binding relation in β.

Again, the requirement that $< q_y, v_{1,y}, \ldots, v_{n,y} >$ is a binding relation in β means that $\{v_{1,y}, \ldots, v_{n,y}\}$ is **exactly** the set of occurrences of variable y bound by occurrence q_y in β, not less, and not more.

Expressions α and β are **alphabetic variants** iff
there is a sequence of expressions $<\alpha_1,\ldots,\alpha_n>$ such that $\alpha_1 = \alpha$ and $\alpha_n = \beta$ and
for every $i<n$: α_i and α_{i+1} are basic alphabetic variants.

THEOREM: if α and β are alphabetic variants then α= β is logivally valid,
true on every model.

The rule about alphabetic variants (many students keep doing this wrong): you change the variable in an occurrence of a variable binding operator plus the occurrences of that variable bound by that occurrence, i.e you change a binding relation.

☠☠☠ You never chance a free variable. That is *not* an alphabetic variant, because the meaning changes (different assignment value).

Alphabetic variant:

yes: $\lambda x.\exists y[R(x,y)] \wedge P(y)$
$\lambda z.\exists u[R(z,u)] \wedge P(y)$ change λx to λz and $\exists y$ to $\exists u$ + variables bound

no: $\lambda x.\exists y[R(x,y)] \wedge P(y)$
$\lambda y.\exists y[R(y,y)] \wedge P(y)$ change λx to λy + variables bound
Reason: the occuring x was bound by λx, and becomes bound by $\exists y$
The second occurrence of y was free and becomes bound by λy

no: $\lambda x.\exists y[R(x,y)] \wedge P(y)$
$\lambda x.\exists y[R(x,y)] \wedge P(z)$
the second occurrence of y was free, replacing it by z, changes the interpretation relative to assignment g

2.5. λ-conversion

TL shares with predicate logic the principle of extensionality:

Extensionality:
Let φ be a expression containing an occurrence of expression α.
Let φ[β/α] be the result of replacing in φ that occurrence of α by β.
Then: (α=β) entails φ = φ[β/α]

Example: Look at the expressions λx.PURR(x) and PURR of type <e,t>, with x ∈ VAR_e.
It is not hard to see, by working out the semantics, that λx.PURR(x) = PURR
Now look at the expression: [λx.PURR(x)](RONYA).
It follows from extensionality that, since λx.PURR(x) = PURR,
[λx.PURR(x)](RONYA) = PURR(RONYA).

So we simplify [λx.PURR(x)](RONYA) with extensionality.

Note my policy of brackets:
[λx.PURR(x)](RONYA) is of course not a TL expression at all.
The correct expression is:

(λx(PURR(x))(RONYA))

Brackets ( ) are introduced in two place operation and (( )) in funcional application,
But I like to leave out brackets where there is no confusion, so I write
λx.PURR(x) instead of λx(PURR(x)).
But then I need to introduce different brackets in λx.PURR(x)(RONYA)
to show the intended constituent structure, I use square brackets around the function for that:
[λx.PURR(x)](RONYA) the result of applying λx.PURR(x) to RONYA.
In other words, the 'readable' version deletes and introduces brackets to suggest the same construction tree but make the (λxβ (α)) structure more visible:

[λx.PURR(x)](RONYA)

End of note.

Extensionality only allows reduction in certain cases.
The most powerful reduction principle of type theory is called λ-conversion
(in untyped λ-calculus it is called β-conversion, for the only reason that it is axiom number two and the axioms are named with greek letters).

λ-conversion

Let $x \in VAR_a$, $\beta \in EXP_b$, $\alpha \in EXP_a$.
Let $\beta[\alpha/x]$ be the result of replacing every free occurrence of x in β by α. Then:

$\lambda x.\beta(\alpha) = \beta[\alpha/x]$ if no variable which is free in α gets bound in $\beta[\alpha/x]$.

λ-conversion is not a rule, but a *fact* about TL.
If the condition stated holds, λ-conversion is valid.
If the condition does not hold, there is no guarantee that the identity holds.

Examples:

⇒

Let $x \in VAR_e$

$[\lambda x.PURR(x)](RONYA) = PURR(RONYA)$
$\lambda x.\beta \quad (\alpha) \quad \beta[\alpha/x]$

We already knew that from extensionality.
This tells us that ronya has the property that you have if you purr iff ronya purrs.

⇒

$[\lambda x.\neg PURR(x)](RONYA) = \neg PURR(RONYA)$
$\lambda x. \quad \beta \quad (\alpha) \quad \beta[\alpha/x]$

This tells us that ronya has the property that you have if you don't purr iff ronya doesn't purr.

⇒

$[\lambda x.PLAY(x) \wedge PURR(x)](RONYA) = PLAY(RONYA) \wedge PURR(RONYA)$
$\lambda x. \quad \beta \quad (\alpha) \quad \beta[\alpha/x]$

This tells us that ronya has the property that you have if you play and purr iff ronya plays and ronya purrs.

Of course, for all these cases, you can check that the identity statement is true by working out the semantics. One example:

$[\![[\lambda x.\neg PURR(x)](RONYA)]\!]_{M,g} = 1$ iff

$[\![[\lambda x.\neg PURR(x)]]\!]_{M,g}([\![(RONYA)]\!]_{M,g}) = 1$ iff

$[\![[\lambda x.\neg PURR(x)]]\!]_{M,g}(F(RONYA)) = 1$ iff

h(F(RONYA)) = 1
where h: D → {0,1} such that for all d ∈ D: h(d) = 1 iff F(PURR)(d) = 0
iff
F(PURR)(F(RONYA)) = 0 iff

$[\![\neg PURR(RONYA)]\!]_{m,g} = 1$

Hence: $[\![[\lambda x.\neg PURR(x)](RONYA)]\!]_{M,g} = [\![\neg PURR(RONYA)]\!]_{M,g}$ for any M,g
And hence: $[\![[\lambda x.\neg PURR(x)](RONYA) = \neg PURR(RONYA)]\!]_{M,g} = 1$ for any M,g

For λ-conversion it doesn't matter what the types of the variables is:

⇒

Let $P \in VAR_{<e,t>}$

$[\lambda P.P(RONYA)](PURR) = PURR(RONYA)$
λP. β (α) β[α/P]

This tells us that PURR is a property in the set of all properties that ronya has iff ronya purrs. Extensionally: PURR is a set in the set of all sets that contain ronya, iff ronya purrs.

Again, showing this semantically:
$[\![[\lambda P.P(RONYA)](PURR)]\!]_{M,g} = 1$ iff

$[\![[\lambda P.P(RONYA)]]\!]_{M,g}[\![(PURR)]\!]_{M,g} = 1$ iff

h(F(PURR)) = 1
where h: $D_{<e,t>} \rightarrow \{0,1\}$ such that for every $X \in D_{<e,t>}$: $[\![P(RONYA)]\!]_{M,g^X_P} = 1$
= h: $D_{<e,t>} \rightarrow \{0,1\}$ such that for every $X \in D_{<e,t>}$: $g^X_P(P)(F(RONYA)) = 1$
= h: $D_{<e,t>} \rightarrow \{0,1\}$ such that for every $X \in D_{<e,t>}$: $X(F(RONYA)) = 1$
= h: $D_{<e,t>} \rightarrow \{0,1\}$ such that for every $X \in D_{<e,t>}$: $X(F(RONYA)) = 1$
iff
F(PURR)(F(RONYA)) = 1 iff
$[\![PURR(RONYA)]\!]_{M,g} = 1$

⇒

[λP.P(RONYA)](λx.¬PURR(x)) = [λx.¬PURR(x)](RONYA)
λP. β (α) β[α/P]
[λx.¬PURR(x)](RONYA) = ¬PURR(RONYA)
λx. β (α) β[α/x]

This tells us that the property that you have if you don't purr is one of the properties in the set of all properties that ronya has iff ronya has the property that you have if you don't purr, and we have seen that, with λ-conversion that is equivalent to: ronya doesn't purr.

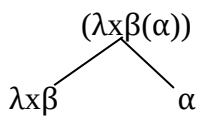
⇒

Let CHASE ∈ $CON_{<e,<e,t>>}$, x,y ∈ VAR_e

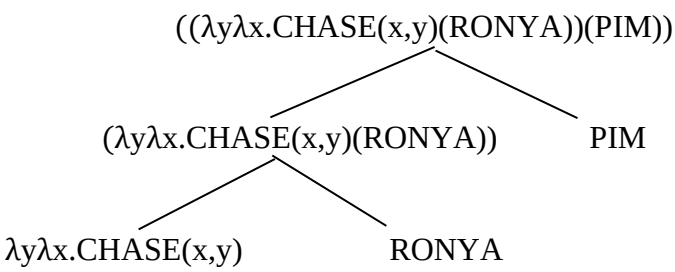
((λyλx.CHASE(x,y)(RONYA))(PIM))

How do you determine what is the proper structure for λ-conversion?

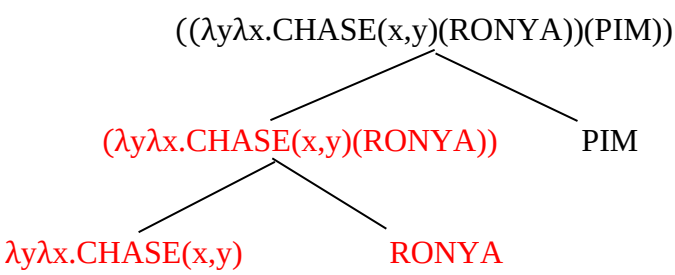
Write (part of) the construction tree of the expression.
Any and only subtrees of the form:



with x ∈ VAR_a and α ∈ EXP_a and (λxβ(α)) a wellformed expression allow λ-conversion (if the condition is satisfied).
Thus we write:



And we see that there is only one subtree of the right kind:



You cannot convert PIM into its sister, because the sister does not start with a lambda, but with a functional application bracket.

Hence, we apply λ-conversion on a sub-expression:

((λyλx.CHASE(x,y)(RONYA))(PIM)) = λx.CHASE(x, RONYA)(PIM)
λy. β (α)

Now λ-conversion becomes possible on λx and PIM and we get:
λx.CHASE(x, RONYA)(PIM) = CHASE(PIM, RONYA)
λx. β (α)

We see that we can unproblematically apply λ-conversion on a sub-expression. Why? Because of extensionality: λ-conversion states that λx.β = β[α/x]. if λx.β is a subexpression of φ, then we can replace by extensionality λx.β in φ by β[α/x].

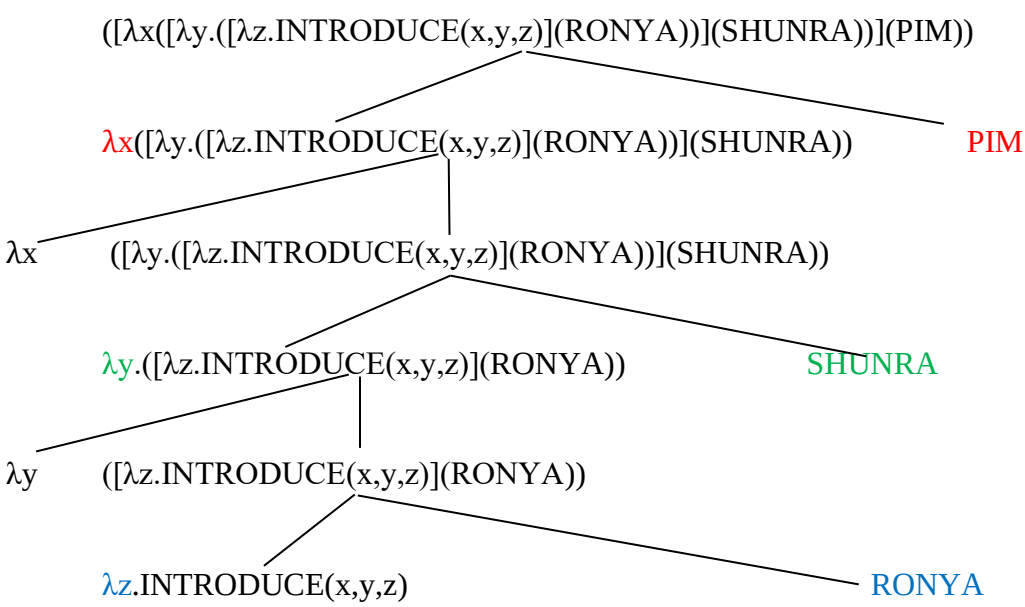
⇒

Let INTRODUCE ∈ $CON_{<e,<e,<e,t>>}$, a three place predicate.

Look at:

([λx([λy.([λz.INTRODUCE(x,y,z)](RONYA))](SHUNRA))](PIM))

Hard to read? write the structure tree:



The construction tree shows that there are three λ-conversions possible in this expression: RONYA for z, or SHUNRA for y or PIM for x.

Question: Which one should we do first?
Answer: That cannot possibly matter.

Why?
Answer: Extensionality.

Showing one way of simpifying the expression:

([λx([λy.([λz.INTRODUCE(x,y,z)](RONYA))](SHUNRA))](PIM)) =
([λx([λy.INTRODUCE(x,y, RONYA)](SHUNRA))](PIM))

([λx([λy.INTRODUCE(x,y,RONYA)](SHUNRA))](PIM)) =
[λy.INTRODUCE(PIM,y, RONYA)](SHUNRA))

[λy.INTRODUCE(PIM,y,RONYA)](SHUNRA)) =
INTRODUCE(PIM, SHUNRA, RONYA)

So:

([λx([λy.([λz.INTRODUCE(x,y,z)](RONYA))](SHUNRA))](PIM)) =
INTRODUCE(PIM, SHUNRA, RONYA)

The condition on λ-conversion:

λx.β(α) = β[α/x] if no variable which is free in α gets bound in β[α/x].

Look at the following situation: Let $x,y \in VAR_e$

λx.∃y[CHASE(x,y)](y) ≠ ∃y[CHASE(y,y)]
λx.β (α)

λx.∃y[CHASE(x,y)](y) expresses that *you* have the property of chasing someone
∃y[CHASE(y,y)] expresses that someone chases himself

Obviously they don't have the same meaning, and the situation violates the condition, because y is free in y, but bound by ∃y in ∃y[CHASE(x,y)][y/x].
So this λ-conversion is not possible.

We can simplify the λx.∃y[CHASE(x,y)](y) by taking an alphabetic variant.

λx.∃y[CHASE(x,y)](y) =
λx.∃z[CHASE(x,z)](y)

λx.∃z[CHASE(x,z)](y) =
∃z[CHASE(y,z)]

∃z[CHASE(y,z)] expresses that *you* chase someone, indeed, equivalent.

Note: Here you will be tempted to reduce:

$\lambda x.\exists y[CHASE(x,y)](y) =$
$\lambda x.\exists y[CHASE(x,y)](z)$

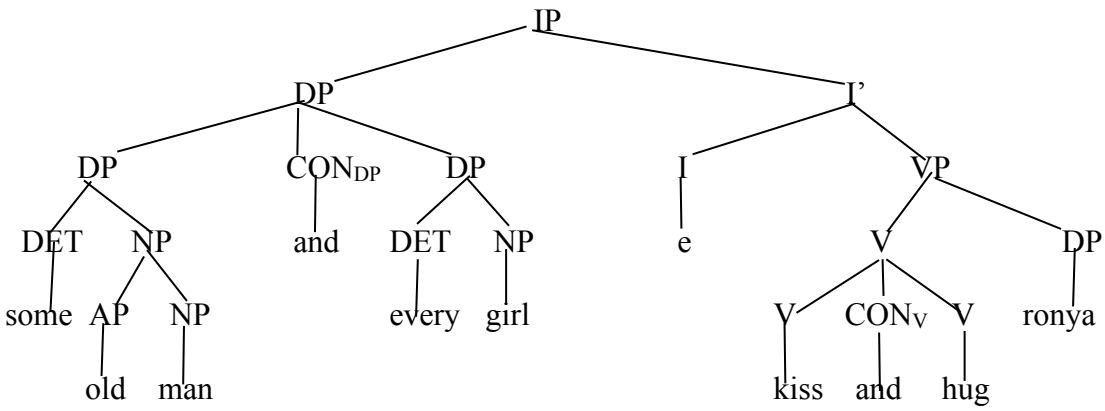
But that is wrong, because it isn't an alphabetic variant: alphabetic variants don't concern free variables but binding relations.

Without comment: there is one more principle characteristic of type logic:

principle of function-identity:
Let $\alpha \in EXP_{<a,b>}$, $x \in VAR_a$. Then
$\alpha = \lambda x.\alpha(x)$
if α doesn't contain a free occurrence of variable x.

2.6. A little grammar.

Let us now make a little grammar that will generate sentence (1) and provide a compositional semantics for it.



Our grammar will generate syntactic structures, and associate with every syntactic structure a corresponding representation in our type logical language, its translation into type logic.

We will do this in a compositional way, that is, we will interpret all the basic expressions, and translate every syntactic operation on syntactic expressions into a semantic operation:

the syntactic operation maps input syntactic structures onto an output syntactic structure, the corresponding semantic operation maps the translations of the input syntactic structures onto the translation of the output syntactic expression.
Since we have already specified the semantic interpretation of the type logical language, in this way, we associate indirectly with each syntactic structure a corresponding semantic interpretation, namely the interpretation of the corresponding type logical translation.

The type logical language is there to make life easy for us.
Of course, the grammar chooses a specific expression as the translation of the syntactic tree. But that expression is only a convenient way of getting at its interpretation: it doesn't matter which expression the grammar chooses, as long as it has that interpretation.
This means that we can, without any problem choose an alphabetic variant if that is more convenient, because it has the same interpretation.
And, even more importantly, we can use the properties of the logical language to write the same information in a simpler way by using λ-conversion.

Thus the grammar will generate pairs <α,β>, where α is a syntactic tree and β is an expression of type logic, the translation of tree α.

We first specify the syntactic categories used in this grammar and their corresponding semantic types.
The grammar will translate any tree with category A as topnode into a type logical expression of the corresponding type.
As we will see, in this little grammar certain categories (DP) have more than one corresponding semantic type.

CATEGORIES AND CORRESPONDING TYPES

NP	→	<e,t>
noun phrases		sets
AP	→	<<e,t>,<e,t>>
adjectives		functions from sets into sets
DET	→	<<e,t>,<<e,t>,t>>
determiners		relations between sets
DP	→	e or <<e,t>,t>
determiner phrases		individuals or sets of sets
V	→	<e,<e,t>>
transitive verb phrases		two-place relations
VP	→	<e,t>
intransitive verb phrases		sets
IP	→	t
sentences		truth values
I	→	<<e,t>,<e,t>>
inflection		functions from sets into sets
CON_V	→	<<e,<e,t>>,<<e,<e,t>>,<e,<e,t>>>
transitive verb phrase connectives		
CON_{DP}	→	<<<e,t>,t>,<<<e,t>,t>,<<e,t>,t>>>
determiner phrase connectives		

The grammar contains the following lexical items:

LEXICAL ITEMS

< NP , GIRL> GIRL $\in CON_{<e,t>}$
|
girl

< NP , MAN> MAN $\in CON_{<e,t>}$
|
man

<AP , $\lambda P \lambda x.P(x) \wedge OLD(x)$> $x \in VAR_e$, $P \in VAR_{<e,t>}$
|
old
$OLD \in CON_{<e,t>}$

<DET , $\lambda Q \lambda P.\forall x[Q(x) \rightarrow P(x)]$> $x \in VAR_e$. $P,Q \in VAR_{<e,t>}$
|
every

<DET , $\lambda Q \lambda P.\exists x[Q(x) \wedge P(x)]$> $x \in VAR_e$, $P,Q \in VAR_{<e,t>}$
|
some

< DP , RONYA> RONYA $\in CON_e$
|
ronya

< V , KISSED> KISSED ∈ $CON_{<e,<e,t>>}$

kissed

< V , HUGGED> HUGGED ∈ $CON_{<e,<e,t>>}$

hugged

<CON_V, λSλRλyλx.R(x,y) ∧ S(x,y)> x,y ∈ VAR_e,
R,S ∈ $VAR_{<e,<e,t>>}$

and

<CON_{DP}, λUλTλP.T(P) ∧ U(P)> P ∈ $VAR_{<e,t>}$
T,U ∈ $VAR_{<<e,t>,t>}$

and

< I, λP.P > P ∈ $VAR_{<e,t>>}$

e

The inflection in this example has no semantic effect, it is interpreted as the identity function at type <e,t>: it maps every set onto itself.

Let us now specify the rules of the grammar. In what follows
< A, A'> stands for a pair consisting of a syntactic tree with topnode A and translation A'.

THE SYNTACTIC AND SEMANTIC RULES

R1. <DET,DET'> + <NP,NP'> ==> < DP , (DET'(NP'))>

DET NP

This rule takes a determiner and a noun phrase and forms a DP, the translation of the DP is the result of applying the translation of the determiner to the translation of the noun.

DET'(NP') ∈ $EXP_{<<e,t>,t>}$

R2. <AP,AP> + <NP,NP'> ==> < NP , (AP'(NP'))>

AP NP

AP'(NP') ∈ $EXP_{<e,t>}$

R3. <V,V'> + <DP,DP'> ==> < VP , (V'(DP'))>

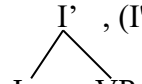
V DP

V'(DP') ∈ $EXP_{<e,t>}$

Note that there is an obvious problem with this rule. It works fine for combining *kissed* with the DP *ronya*, but the syntax also allows us to combine *kissed* with the DP *every girl*, and the semantics doesn't work for that case because the types don't match: in that case $V' \in EXP_{<e,<e,t>>}$ and $DP' \in EXP_{<<e,t>,t>}$, hence V'(DP') is not wellformed.

Since we are only dealing with an example here, we assume that we only use this rule with DPs that are proper names. We will come back to this problem shortly.

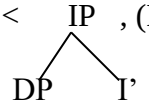
R4. <I,I'> + <VP,VP'> ==> < I' , (I'(VP'))>



$I'(VP') \in EXP_{<e,t>}$

(I' is read I-bar in the syntax, but I-apostrophe in the semantics.)

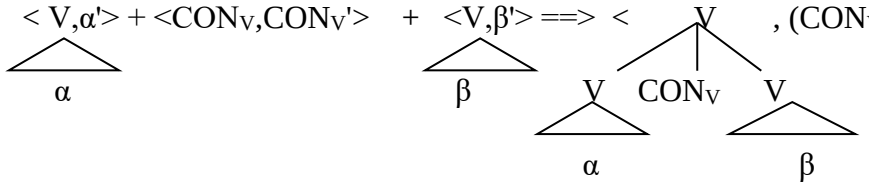
R5. <DP,DP'> + <I',I''> ==> < IP , (DP'(I''))>



$DP'(I'') \in EXP_t$

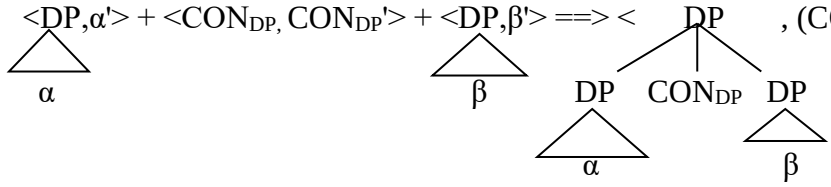
This rule has the inverse problem from the previous one. This time the rule works fine if we combine the DP *every girl* with the I' *walked*, because $DP' \in EXP_{<<e,t>,t>}$ and $I'' \in EXP_{<e,t>}$. But the syntax allows us to combine *ronya* with *walked* as well, and in that case DP'(I'') is not well formed, because $DP' \in EXP_e$ and $I'' \in EXP_{<e,t>}$. Again, we will only be concerned with sentence (1) and assume that the rule doesn't apply to proper names.

R6. $< V,\alpha'> + <CON_V,CON_V'> + <V,\beta'> ==> <$ V , $(CON_V'(\beta'))(\alpha')>$



$(CON_V'(\beta))(\alpha) \in EXP_{<e,<e,t>>}$

R7. $<DP,\alpha'> + <CON_{DP}, CON_{DP}'> + <DP,\beta'> ==> <$ DP , $(CON_{DP}'(\beta'))(\alpha')>$



$(CON_{DP}'(\beta))(\alpha) \in EXP_{<<e,t>,t>}$

This is the grammar. We can now give a derivation for sentence (1):

(1) Some old man and every girl kissed and hugged Ronya.

In this derivation, we will see the usefulness of λ-conversion in reducing translations to readable ones.

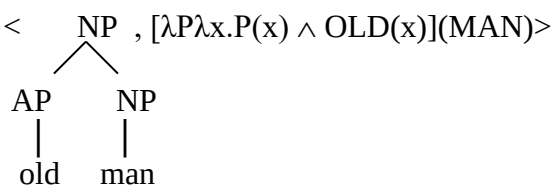
THE DERIVATION

We start with the lexical items for *old* and *man*:

<AP , λPλx.P(x) ∧ OLD(x)>
|
old

< NP , MAN>
|
man

R2 applies to these and forms:



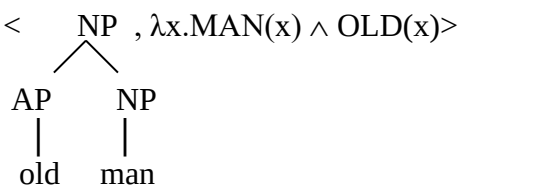
The translation is:

[λPλx.P(x) ∧ OLD(x)](MAN)

This expression can be reduced by λ-converting MAN for P:

[λPλx.P(x) ∧ OLD(x)](MAN) =
λx.MAN(x) ∧ OLD(x)
the property that you have if you're a man and you're old.

Thus, after reduction, the grammar produces:

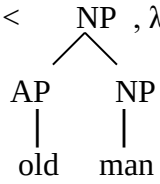


Now we take the lexical item for *some*:

<DET , λQλP.∃x[Q(x) ∧ P(x)]>
|
some

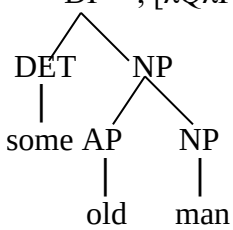
and the result we got for *old man*:

<NP , λx.MAN(x) ∧ OLD(x)>



Rule R1 applies to these and forms a noun phrase:

<DP , [λQλP.∃x[Q(x) ∧ P(x)]](λx.MAN(x) ∧ OLD(x))>



Let us reduce the resulting translation:

[λQλP.∃x[Q(x) ∧ P(x)]](λx.MAN(x) ∧ OLD(x))

First, just to make the formula more readable, lets take alphabetic variants and replace λx by λz:

[λQλP.∃x[Q(x) ∧ P(x)]](λx.MAN(x) ∧ OLD(x)) =

[λQλP.∃x[Q(x) ∧ P(x)]](λz.MAN(z) ∧ OLD(z))
[λQ--------Q-------------](α)
  <e,t> <e,t>

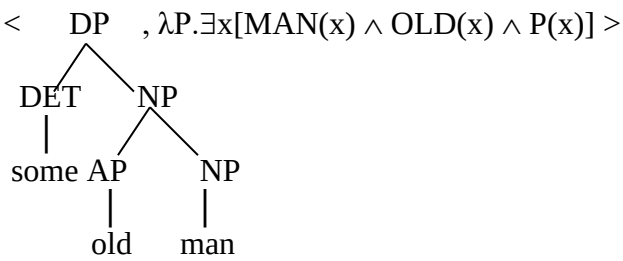
λz.MAN(z) ∧ OLD(z) ∈ $EXP_{<e,t>}$, hence we convert it in for λQ and it gets substituted for variable Q:

[λQλP.∃x[Q(x) ∧ P(x)]](λz.MAN(z) ∧ OLD(z)) =
λP.∃x[[λz.MAN(z) ∧ OLD(z)](x) ∧ P(x)]
  [λz.-------z-------------z-](α)

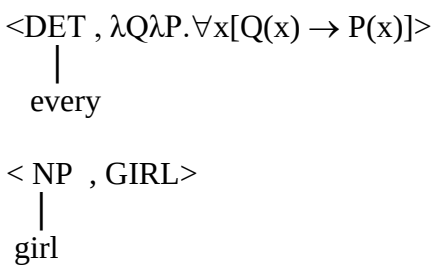
On this expression, we can do once more λ-conversion, converting variable x for λz.
In this way x gets substituted for both occurrences of z:

λP.∃x[[λz.MAN(z) ∧ OLD(z)](x) ∧ P(x)] =
λP.∃x[MAN(x) ∧ OLD(x) ∧ P(x)]
the set of properties that some old man has

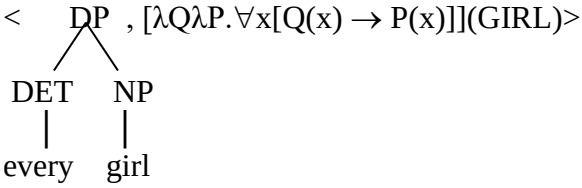
Thus we get:



Next we take the lexical items for *every* and *girl*:



Once again, rule R1 applies to these and forms a noun phrase:



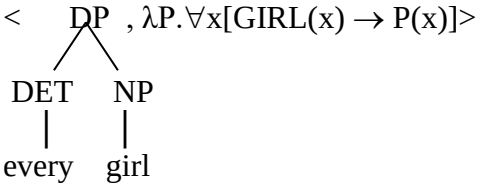
We reduce the translation by λ-conversion:

[λQλP.∀x[Q(x) → P(x)]](GIRL)
[λQ--------Q----------](α)

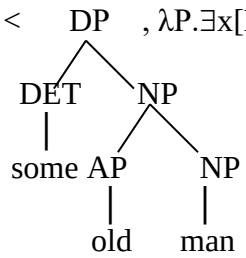
GIRL gets converted in for λQ, we get:

[λQλP.∀x[Q(x) → P(x)]](GIRL) =
λP.∀x[GIRL(x) → P(x)]
the set of properties that every girl has

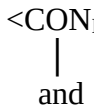
Hence we get:



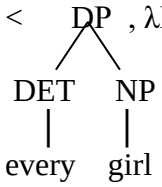
We next take *some old man*:



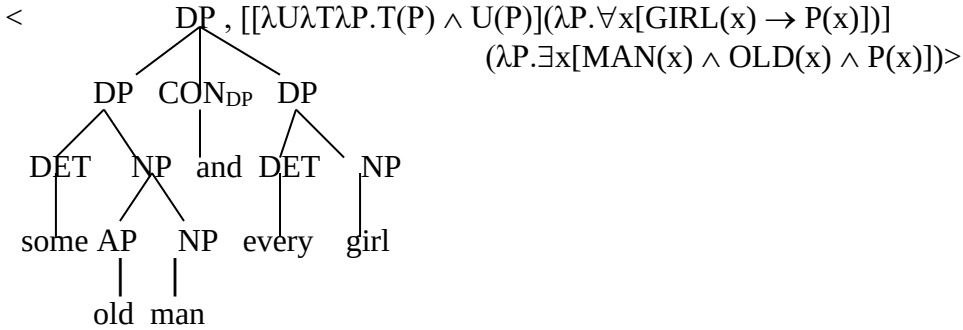
and the lexical item of *and* as an DP-connective:



and *every girl*:



And we apply rule R7 to these. This gives:



Let us reduce the translation:

[[λUλTλP.T(P) ∧ U(P)](λP.∀x[GIRL(x) → P(x)])]
(λP.∃x[MAN(x) ∧ OLD(x) ∧ P(x)])

Let us first take an alphabetic variant, so that we won't get confused with our variables. We replace λP in *every girl*' by λQ, and similarly in *some old man*'.

[[λUλTλP.T(P) ∧ U(P)](λP.∀x[GIRL(x) → P(x)])]
(λP.∃x[MAN(x) ∧ OLD(x) ∧ P(x)]) =

[[λUλTλP.T(P) ∧ U(P)](λQ.∀x[GIRL(x) → Q(x)])]
(λQ.∃x[MAN(x) ∧ OLD(x) ∧ Q(x)])

In this expression U is variable of type <<e,t>,t>,
λQ.∀x[GIRL(x) → Q(x)] is an expression of type <<e,t>,t>, hence we can apply λ-conversion. λU disappears, and λQ.∀x[GIRL(x) → Q(x)] gets substituted for U in λTλP.T(P) ∧ U(P). The rest stays as is. So we get:

[[λUλTλP.T(P) ∧ U(P)](λQ.∀x[GIRL(x) → Q(x)])]
(λQ.∃x[MAN(x) ∧ OLD(x) ∧ Q(x)]) = [by λ-conversion]
v
[λTλP.T(P) ∧ [λQ.∀x[GIRL(x) → Q(x)]](P)] (λQ.∃x[MAN(x) ∧ OLD(x) ∧ Q(x)])

In this expression we can λ-convert P for λQ in [λQ.∀x[GIRLY(x) → Q(x)]](P), hence we get:
[λTλP.T(P) ∧ [λQ.∀x[GIRL(x) → Q(x)]](P)] (λQ.∃x[MAN(x) ∧ OLD(x) ∧ Q(x)]) =

[λTλP.T(P) ∧ ∀x[GIRL(x) → P(x)]] (λQ.∃x[MAN(x) ∧ OLD(x) ∧ Q(x)])

T is a variable of type <<e,t>,t>, hence we can λ-convert
λQ.∃x[MAN(x) ∧ OLD(x) ∧ Q(x)] for λT: λT disappears, and
λQ.∃x[MAN(x) ∧ OLD(x) ∧ Q(x)] gets substituted for T in
λP.T(P) ∧ ∀x[GIRL(x) → P(x)]. Hence, we get:

[λTλP.T(P) ∧ ∀x[GIRL(x) → P(x)]] (λQ.∃x[MAN(x) ∧ OLD(x) ∧ Q(x)]) =

λP.[λQ.∃x[MAN(x) ∧ OLD(x) ∧ Q(x)]](P) ∧ ∀x[GIRL(x) → P(x)]]

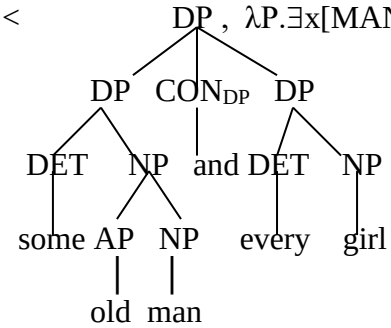
One more λ-conversion converts P for λQ, giving:

λP.[λQ.∃x[MAN(x) ∧ OLD(x) ∧ Q(x)]](P) ∧ ∀x[GIRL(x) → P(x)]] =

λP.∃x[MAN(x) ∧ OLD(x) ∧ P(x)] ∧ ∀x[GIRL(x) → P(x)]
The set of properties that some old man has and that every girl has as well.

Thus we get:

<DP , λP.∃x[MAN(x) ∧ OLD(x) ∧ P(x)] ∧ ∀x[GIRL(x) → P(x)]>



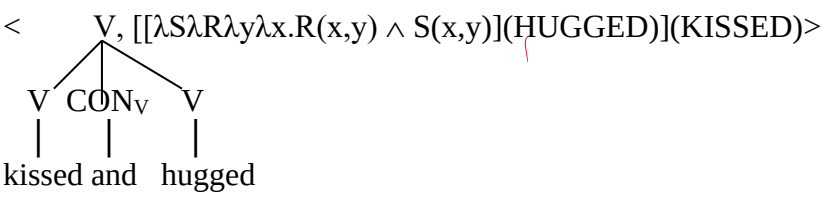
Next we take the lexical items for *kissed* and *and* as a TV-connective and *hugged*:

< V , KISSED>
|
kissed

<CON$_V$, λSλRλyλx.R(x,y) ∧ S(x,y)>
|
and

< V , HUGGED>
|
hugged

Rule R6 applies to these and gives:



S is a variable of type <e,<e,t>>, we convert HUGGED in and get:

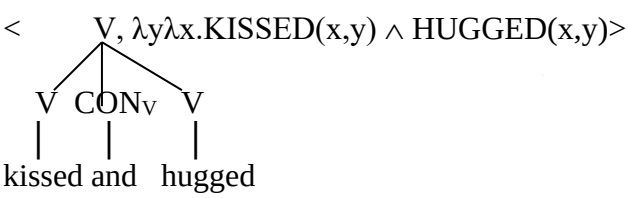
[[λSλRλyλx.R(x,y) ∧ S(x,y)](HUGGED)](KISSED) =

[λRλyλx.R(x,y) ∧ HUGGED(x,y)](KISSED)

Converting KISSED for λR gives:

λyλx.KISSED(x,y) ∧ HUGGED(x,y)
The relation that me and you stand in if you kissed and hugged me

Thus we get:

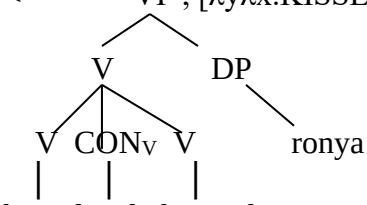


Next rule R3 combines this and the lexical item for *ronya*,

< DP , RONYA>
|
ronya

into:

< VP , [λyλx.KISSED(x,y) ∧ HUGGED(x,y)](RONYA)>



h

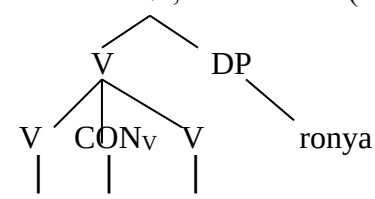
λ-conversion converts RONYA for λy, RONYA gets substituted for both occurrences of variable y:

[λyλx.KISSED(x,y) ∧ HUGGED(x,y)](RONYA) =

λx.KISSED(x,RONYA) ∧ HUGGED(x,RONYA)
The property that you have if you kissed Ronya and you hugged Ronya.

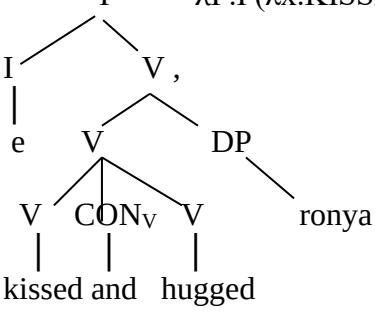
Hence we get:

< V , λx.KISSED(x,RONYA) ∧ HUGGED(x,RONYA)>

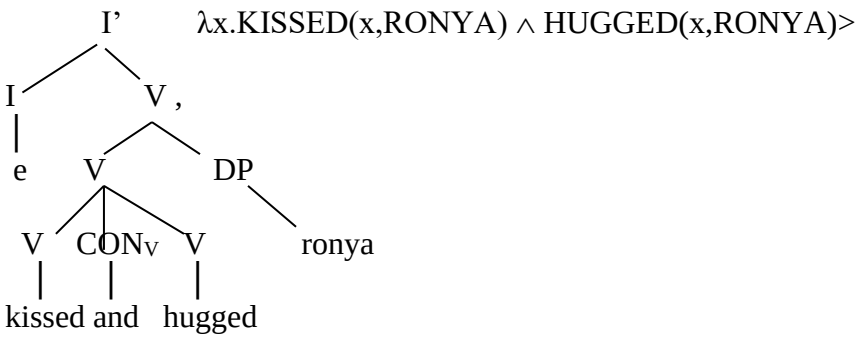


Next, R4 brings in the I:

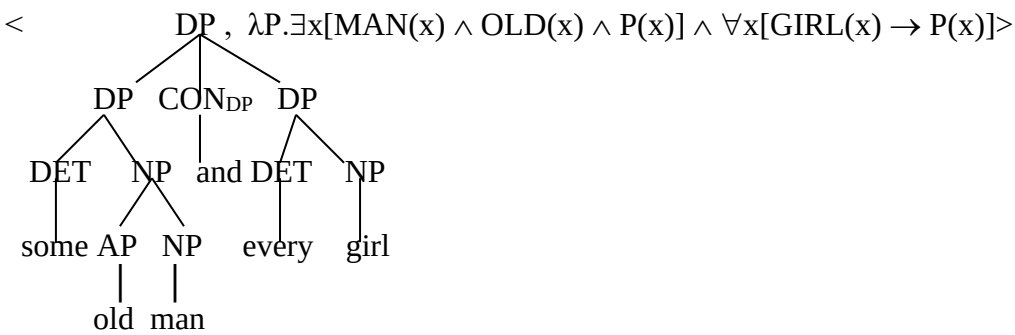
I' λP.P(λx.KISSED(x,RONYA) ∧ HUGGED(x,RONYA))>



With λ-conversion λP.P(λx.KISSED(x,RONYA) ∧ HUGGED(x,RONYA)) reduces to:
λx.KISSED(x,RONYA) ∧ HUGGED(x,RONYA)
So we get:

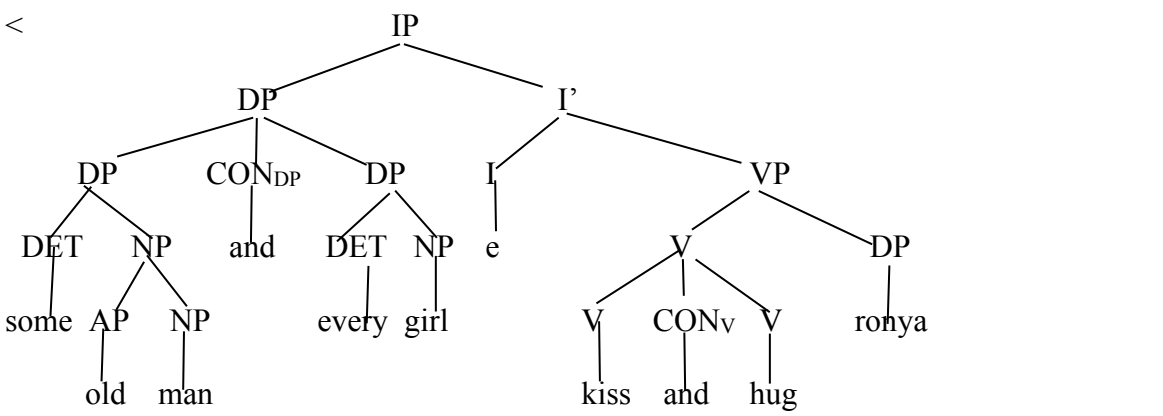


Now we take the DP *some old man and every girl* that we have built up:



and the I’ *kissed and hugged Ronya* that we have just built up.

Rule R5 combines the two and gives:



[λP.∃x[MAN(x) ∧ OLD(x) ∧ P(x)] ∧ ∀x[GIRL(x) → P(x)]
(λx.KISSED(x,RONYA) ∧ HUGGED(x,RONYA))>

Again, to avoid collision of variables we change λx to λz:

[λP.∃x[MAN(x) ∧ OLD(x) ∧ P(x)] ∧ ∀x[GIRL(x) → P(x)]]
(λx.KISSED(x,RONYA) ∧ HUGGED(x,RONYA)) =

[λP.∃x[MAN(x) ∧ OLD(x) ∧ P(x)] ∧ ∀x[GIRL(x) → P(x)]]
(λz.KISSED(z,RONYA) ∧ HUGGED(z,RONYA))

We convert λz.KISSED(z,RONYA) ∧ HUGGED(z,RONYA) for λP. It gets substituted for both occurrences of variable P:

[λP.∃x[MAN(x) ∧ OLD(x) ∧ P(x)] ∧ ∀x[GIRL(x) → P(x)]]
(λz.KISSED(z,RONYA) ∧ HUGGED(z,RONYA)) =

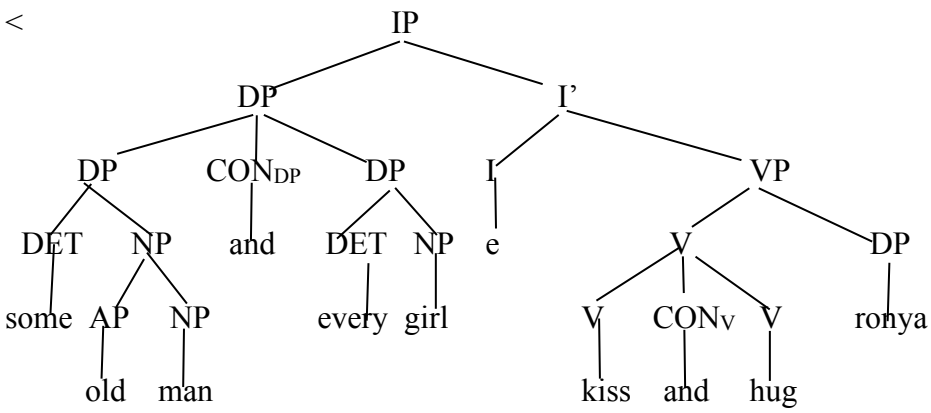
∃x[MAN(x) ∧ OLD(x) ∧ [λz.KISSED(z,RONYA) ∧ HUGGED(z,RONYA)](x)] ∧
∀x[GIRL(x) → [λz.KISSED(z,RONYA) ∧ HUGGED(z,RONYA)](x)]

There are two λ-convertions left: we convert x for λz in the first conjunct, it gets substituted for both occurrences of variable z; and we do the same in the second conjunct. The result is:

∃x[MAN(x) ∧ OLD(x) ∧ [λz.KISSED(z,RONYA) ∧ HUGGED(z,RONYA)](x)] ∧
∀x[GIRL(x) → [λz.KISSED(z,RONYA) ∧ HUGGED(z,RONYA)](x)] =

∃x[MAN(x) ∧ OLD(x) ∧ KISSED(x,RONYA) ∧ HUGGED(x,RONYA)] ∧
∀x[GIRL(x) → KISSED(x,RONYA) ∧ HUGGED(x,RONYA)]

Hence, we finally derive:



∃x[MAN(x) ∧ OLD(x) ∧ KISSED(x,RONYA) ∧ HUGGED(x,RONYA)] ∧
∀x[GIRL(x) → KISSED(x,RONYA) ∧ HUGGED(x,RONYA)] >
Some old man kissed Ronya and hugged Ronya and every girl kissed Ronya and hugged Ronya.

The leaves of the syntactic tree form sentence (1), hence the grammar generates that sentence with the above syntactic structure and translation.

Note that at every stage of the derivation, we can semantically interpret the translation of the constituent that the grammar derives at that stage. The translation process, and hence the interpretation, is completely compositional.

Let us define truth for sentences relative to a grammatical analysis:

> Let φ be a string.
> <A,A'> is a **grammatical analysis of** φ iff <A,A'> is a pair consisting of a syntactic tree with topnode A and leaves φ and A' is a type logical expression of a type corresponding to A, and the grammar generates <A,A'>.

Let φ be a sentence (as a string) and <IP,IP'> a grammatical analysis of φ.

We define:

> φ is true relative to <IP,IP'> in a model M relative to an assignment function g iff $[\![IP']\!]_{M,g}=1$

Hence we predict that sentence (1) is true in a model
M = <D,F> relative to an assignment function g iff:

$[\![\exists x[MAN(x) \wedge OLD(x) \wedge KISSED(x,RONYA) \wedge HUGGED(x,RONYA)] \wedge$
$\forall x[GIRL(x) \rightarrow KISSED(x,RONYA) \wedge HUGGED(x,RONYA)]]\!]_{M,g}=1$ iff

for some $d \in F(MAN)$: $d \in F(OLD)$ and
$\langle d,F(RONYA)\rangle \in F(KISSED)$ and $\langle d,F(RONYA)\rangle \in F(HUGGED)$
and for every $d \in F(GIRL)$:
$\langle d,F(RONYA)\rangle \in F(KISSED)$ and $\langle d,F(RONYA)\rangle \in F(HUGGED)$

These are of course the right truth conditions.

In the derivation, I have at every stage done λ-conversions as much as possible.
I did not have to do that, of course. Since λ-conversions preserve meaning we can decide at any stage to do or not do a particular λ-conversion.
Without doing any λ-conversions, we would have generated sentence (1) with translation:

$[\ [\lambda U\lambda T\lambda P.T(P) \wedge U(P))](\lambda Q\lambda P.\forall x[Q(x) \rightarrow P(x)](GIRL))$
$(\lambda Q\lambda P.\exists x[Q(x) \wedge P(x)](\lambda P\lambda x.P(x) \wedge OLD(x)(MAN)))\ ]$
$([\lambda S\lambda R\lambda y\lambda x.R(x,y) \wedge S(x,y)](HUGGED)(KISSED)(RONYA))$

This shows the compositional structure, and allows all the λ-conversions we did along the way.