CHAPTER FOUR: TYPE SHIFTING

4.1. Montague's fixed type assumption

So far we have adopted Montague's Fixed Type Assumption:

> **Fixed Type Assumption**.
> The grammar associates with every syntactic category a unique semantic type.
> All syntactic structures of that syntactic category translate as expressions
> of the corresponding semantic type.

We have seen that in several cases the fixed type assumption leads us to adopt translations for expressions that are more complicated than one would initially assume necessary.
For instance, the assumption that proper names denote individuals, and hence are expressions of type e seems natural.
We have also seen that we cannot assume that DPs like *every girl* are of type e.
Since arguably proper names are DPs as well, the Fixed Type Assumption forces us to choose between these types e and <<e,t>,t> as the type associated with DP,
and since for semantic reasons we cannot choose e, it has to be <<e,t>,t>.

This means that we are forced to give up the nice and simple analysis of proper names.

Fortunately, there is a translation for proper names at the type of generalized quantifiers that will work as well: λP.P(RONYA) for *ronya*.

Such considerations have effects in the grammar at many places.

It also seems to be a natural initial assumption that a transitive verb like *hug* denotes a relation between individuals of type <e,<e,t>>.
But the above considerations tell us that the type corresponding to TV will have to be at least <<<e,t>,t>,<e,t>>. We gave *hug* the translation:

> λTλx.T(λy.HUG(x,y)).

Now, of course, we do get an elegant and general theory here, but it does come at a price:
Montague's strategy is justifiably known as:

> **Generalize to the worst case**:
> Because the analysis of some expression of category C requires a more complicated type than other expressions of that category,
> all expressions of that category are given the more complicated type.

One of the disadvantages of this approach is that, when you are analyzing the simpler cases, you do get the feeling that you are doing for generality sake what you could have done in a much simpler way.
So, for generality sake, we decide to apply the complicated translation of *hug* to the complication translation of *ronya*:

> [λTλx.T(λy.HUG(x,y))](λP.P(RONYA))

while for that particular case, everything would have worked as well, had we decided not to use the complicated translations, but just apply the simple translation HUG to the simple translation RONYA:

HUG(RONYA)

Thus, for generality sake, we're doing a lot of extra work, and introduce a lot of unnecessary unreadable translations.

Moreover, things don't stop here.
We have translated verbs like *hug* and *find* as relations of type <<<**e,t>,t>**,<e,t>>.
But *hug* and *find* are extensional relations between individuals,
and, as we have seen in Foundations, there are also verbs, like *seek*, which cannot be analyzed as extensional relations between individuals:

Anna seeks a unicorn

Montague introduced IL, Intensional type Logic. We will discuss this in the next chapter.
At this point we can say that Montague introduces types for intensional entities:
as we will see, for every TL type a there is in IL also an intensional type <s,a>.
While so far we can succesfully regard *find* as a relation of type <<<e,t>,t>,<e,t>>,
the intensionality of *seek* requires an intensional type, for instance <<**s,<<e,t>,t>>**,<e,t>>
which means that *seek* would denote a relation between an individual and an intensional object.

We will see in the next chapter how this works (the basis of it has already been discussed in Foundations), but here I draw attention to what it means for Montague's grammar.

Both *seek* and *find* and transitive verbs of category V, verbs, moreover, that can be conjoined: *Anna sought and found a unicorn*.

But now the Fixed Type Assumption says that we must interpret both *seek* and *find* at the same type, and this will have to be the higher intensional type.
So we must find for *find* an interpretation at the intensional type <<**s,<<e,t>,t>>**,<e,t>>,
even though we would be happy to have it at type <<<e,t>,t>,<e,t>>.

Finding the right interpretation is not a problem:
Once we have formulated the intensional type theory,
we will be able with backwards λ-conversion derive the correct interpretation at the higher type (as Montague incorporated into his theory).

So, while we can resolve this problem to our satisfaction, we are once more only doing this work for generality sake:
Nothing in the meaning of *find* forces this higher interpretation:
translate *find* this high for the sake of *seek*.

This going higher and higher may make you a bit queazy.
What guarantee do we have that this process will ever stop?
Maybe we have overlooked another expression of the same category V, or of the category

DP, which Vs take as argument that have to be interpreted at a yet higher type.
That will force yet another change in the whole grammar: all our interpretations will have to be changed to the higher type.

Indeed, that is just what happens in Montague's work.
Remember the discussion of individual concepts and the Trainer paradox in Foundations, the invalidity of the pattern:

> *The trainer is Michels,*
> *the trainer changes,*
> *hence Michels changes.*

We saw that DP *the trainer* must sometimes be analyzed as an expression denoting an individual concept, a function from worlds to individuals.
For Montague, that is an expression of type <s,e>, rather than e.

But if *some* DPs are interpreted at type <s,e>, then *all* DPs must be interpreted at type <s,e>, by the Fixed Type Assumption.
And DPs were *not* interpreted as type e, but at <<e,t>,t>.
To deal with *the trainer*, we need to assume that the interpretation type is really <<**<s,e>,**t>,t>.
And in fact, to be able to be the complement of an intensional verb, the type must even be intensional: <s,<<**<s,e>,**t>,t>>.

And this means that *seek* is a relation between an intensional entity of this type and an individual concept: <<s,<<**<s,e>,**t>,t>>, <**<s,e>,**t>>.
Or the intension of that.
And that means that a sentence like

> *Ronya finds pim*

involves an interpretation of *ronya* and of *pim* of type <s,<<<s,e>,t>,t>>,
and *find* of type <<s<<s,<<<s,e>,t>,t>>, <<s,e>,t>>,
and the interpretations must be tailored in such a way that the derived sentence reduces with λ-conversion to FIND(RONYA,PIM) with FIND ∈ $CON_{<e,<e,t>>}$.

And indeed, it is a major achievement of Montague's PTQ that this is exactly what happens there.

But it shows the problem:
There is no guarantee that the process will stop even here,
and it is not clear that there *is* a worst type to generalize to.
That makes the theory very instable.

4.2. Flexible types, type shifting and type driven translation

An alternative way of setting up the grammar was initiated by Barbara Partee and Mats Rooth, in Partee and Rooth 1983.
While some earlier work on categorial grammar, notably by Lambek and by Geach, can be regarded as foreshadowing some of their ideas, the Partee and Rooth paper properly stands at the beginning of the theory of **type shifting rules**.

Partee and Rooth give up the Fixed Type Assumption and replace it by what could be called the Flexible Type Assumption:

> **Flexible Type Assumption**: the grammar associates with every syntactic category C a set of types.

With the Fixed Type Assumption out of the way, Partee and Rooth adopt a methodological principle which is practically the opposite of Generalize to the Worst Case:

> **Methodological guide for interpretations**:
> In setting up the grammar, take the simplest cases as your guide and try to give every expression as simple a translation as you can.
> *De facto* this means: interpret expression at *as low a* type as you can.
>
> To adopt to this context a guideline that Emmon Bach suggested for syntax:
>
> **The Partee-Bach's guideline:**
> Interpret everything as low as possible, but not so low that you regret it later.

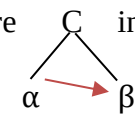
The methodological guide allows different expressions of the same syntactic category to be interpreted at different types, e.g. *Ronya* and *Every cat*, or *find* and *seek*.
It also allows for the possibility of interpreting one expression at different types.

The latter happens in the context of **type driven translation.**

The key ideas of type driven translation are clearly present in Partee and Rooth 1983, approach was explicitly formulated in Klein and Sag 1985, and has been explored in much other work, like Partee 1987, in Maria Bittner's work (e.g. Bittner 1993), and in my own work (Landman 2000, 2004), and many others.

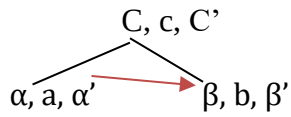
I will call the general approach: **Type Mismatch Grammar**.

Type mismatch grammar

Assume, as before, a binary structure 

interpreted via functional application, say, of α to β.

Assume, with the methodological guide that α and β are interpreted as α' and β' at the lowest type.
We may even know independently the lowest type for C as well, and the interpretation of C at that type.



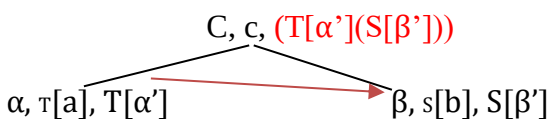
What happens if there is **type mismatch,** i.e. (α(β)) = C' doesn't hold,
in particular, if (α(β)) is not a wellformed expression of type c?

Montague:
The grammar crashes, and you have to assign new, higher types to α and/or β and compute with backward λ-conversion what the new interpretations of α and β at the new types are.
i.e. you set up the grammar in such a way that this doesn't happen.

Type Mismatch grammar:
Type mismatch is completely normal and to be expected.
Type mismatch is not a problem, if the type mismatch can be resolved.
The grammar contains a type mismatch resolution mechanism in the form of a set of **Type Shifting Operations**.

> A **type shifting operation** T shifts interpretation α at type a to interpretation T[α] at type T[a].

A resolution of the type mismatch would take the form:



α' cannot apply to β' to given a wellformed expression of type c.
But the type shifting theory contains two rules T and S, such that T[α'], the result of shifting α' with T applies to S[β'], the result of shifting β' with S to to give a wellformed expression of type c.
(We will include ID: a → a, for any type a among our type shifting rules, so this actually allows also doing only one shift.)

So this is the general idea:

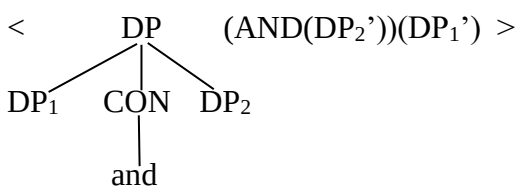
Type Mismatch Grammar
-Interpret everything as low as you can.
-Try to get away which combining expressions into complexes without doing anything special.
-You do something special when you get a type mismatch:
In that case you try to resolve the type mismatch with type shifting operations.
If you can, that is the interpretation you derive.
If you can't, *then* the derivation crashes.

I have used functional application as an example, but the same would be true for other semantic operations, like the operation of coordination, or function composition.

Just to get the flavour of it, let us look at *ronya and every kitten*.

we have assumed:

ronya → RONYA of type e
every kitten → λP.∀x[KITTEN(x) → P(x)] of type <<e,t>,t>
and → AND where AND = λUλTλP.T(P) ∧ U(P)



This gives:

AND(λP.∀x[KITTEN(x) → P(x)]) =
λTλP.T(P) ∧∀x[KITTEN(x) → P(x)]

But:

λTλP.T(P) ∧∀x[KITTEN(x) → P(x)](RONYA)

is not wellformed, because the function λTλP.T(P) ∧∀x[KITTEN(x) → P(x)]
is of type <<**<e,t>,t**>,<<e,t>,t>> but the argument RONYA is of type **e**.

So we have a type mismatch.
We check whether there is a type shifting principle in the set that belongs to our grammar that resolves the type mismatch.

And yes, there is:

Argument Lift: LIFT: e → <<e,t>,t>
for all α ∈ EXP_e: LIFT[α] = λP.P(α)

[P a variable that doesn't occur free in α]

So we **resolve** the infelicity of

λTλP.T(P) ∧∀x[KITTEN(x) → P(x)](RONYA)

by lifting RONYA to λP.P(RONYA), and applying the function to (an alphabetic variant of) that:

λTλP.T(P) ∧∀x[KITTEN(x) → P(x)](LIFT[RONYA])) =
λTλP.T(P) ∧∀x[KITTEN(x) → P(x)](λQ.Q(RONYA))) =
λP.[λQ.Q(RONYA)](P) ∧ ∀x[KITTEN(x) → P(x)] =
λP.P(RONYA)) ∧ ∀x[KITTEN(x) → P(x)]

The huge advantage comes in *ronya purrs*:
We assumed that the verb was function on the subject,
and *purr* is interpreted as PURR ∈ $CON_{<e,t>}$.
That means that we can revert back to the simplest grammatically analysis of *ronya purrs*

ronya purrs → PURR(RONYA)

No type shifting is needed, because the types fit perfectly to the lowest interpretations.

4.3. A semantically interpreted X'-grammar with type shifting

Up to now, I have basically ignored the fact that most of the syntactic rules that I discussed in the context of example grammars have essentially the same form, or the fact that they have essentially the same semantic interpretation.

Moreover, I have not engaged in any attempt to separate what might be universal, common to all languages, from what must be specific, particular to the language in question, English.

As a consequence, many things are stipulated again and again in each rule, that might more economically be stipulated once, i.e. the rules have a lot of redundancy in them.

As is well known, a lot of linguistic thought has gone into developing a format for grammatical theory which does not contain this kind of redundancy.

In syntax, X-bar theory is an obvious example of an attempt of providing (part of) a framework for syntax which tries to be non-redundant and inspires you to think about how to localize and minimize the bits of grammar that must be particular.

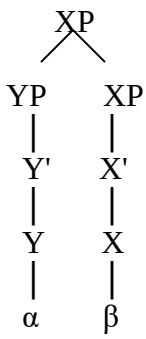
We need not here enter into the debate whether there is an empirical issue at stake here or not (most X-bar syntacticians will tell you that this is obviously so, but see Pullum 199?).

Nevertheless, the heuristic advantage of a format of grammar that separates the universal from the particular is uncontested.
That is, a redundant grammar format may contain the same information as a non-redundant format,
there may even be a procedure extracting the universals or the particulars from the redundant grammar,
but if the material is presented from the start in the form of a general set of constraints on structure and interpretation plus a set of particular constraints and options for the language, or language group, in question,
the format can force or inspire us from the start to think about universals and particulars, which, as we all agree, is a linguistic virtue.

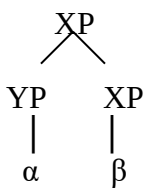
X-bar syntax

Let us take X-bar theory as an example. (And note, it's only an example, I'm not arguing for a particular version of X-bar theory, or even X-bar theory itself, here.)

Let us make the following universal assumptions about syntax.
Categories come in bar-levels X, X', XP, where XP dominates X' and X' dominates X.
Lexical items are sitting under X.
We will directly make the assumption that unary branches are pruned to the highest category. So, our X'-theory may allow the following tree:



By the Principle of Saving the Rainforest, this tree is pruned to:

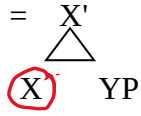


The set of categories CAT will include the categories A and N of adjectives and nouns.

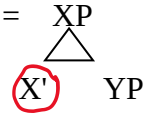
We define three syntactic operations, **complementation**, **specification**, and **modification**: operations that map pairs of categories onto syntactic triangles of the following form:

Let X, Y ∈ CAT

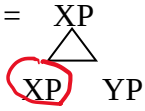
COMP[X,Y] =



SPEC[X,Y] =



MOD[X,Y] =



In these triangles, we call the category at the left bottom corner of the triangle the **head**, hence each of these operations specifies a head.

TRIANGLE = {COMP[X,Y], SPEC[X,Y], MOD[X,Y]: X,Y ∈ CAT}

What we will **base** the syntax on is the set of pairs <α,π>, where α is a triangle and π is one of the symbols in {*l*,*r*}: head-left, head-right:

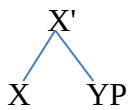
BASE= { <α,π>: α ∈ TRIANGLE and π ∈ {*l*,*r*} }

A syntax for language L is a subset $SYN_L \subseteq$ BASE

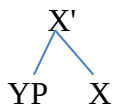
We assume the following realization principle relating objects in SYN_L to trees:

REALIZATION:

If <COMP[X,Y],*l*> ∈ SYN_L then L licences a complementation tree with the head X as left daugher:



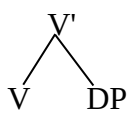
If <COMP[X,Y],*r*> ∈ SYN_L then L licences a complementation tree with the head X as right daugher:



Similarly for the other triangles.
Finally, for triangle α and language L we introduce the following notation:

$\alpha =_L l$ iff $\langle\alpha,l\rangle \in L \wedge \langle\alpha,r\rangle \notin L$
$\alpha =_L r$ iff $\langle\alpha,r\rangle \in L \wedge \langle\alpha,l\rangle \notin L$
$\alpha =_L \perp$ iff $\langle\alpha,l\rangle \in L \wedge \langle\alpha,r\rangle \in L$
$\alpha =_L 0$ iff $\langle\alpha,l\rangle \notin L \wedge \langle\alpha,r\rangle \notin L$

Thus, for categories V and D, we may specify that COMP[V,D] $=_{ENGLISH}$ *l* and this means that in English, verbs take DP complements, and take DP complements to their right, not to their left. This, in its turn means that the following tree is realized in English:



In general,
if you specify: R[X,Y] $=_L$ *l*, this means the head is left
if you specify: R[X,Y] $=_L$ *r*, this means the head is right

if both are specified, the head can be either
if neither is specified, X doesn't relate to Y by R

It will in fact be usuful to specify, for $\alpha \in$ {COMP, SPEC, MOD} and L a language:

$\alpha_L = \{<X,Y>: \alpha[X,Y] \neq_L 0\}$

This allows us to specify, for instance:

$<V,D> \in COMP_{ENGLISH}$
$\forall X,Y \in CAT: COMP[X,Y] =_{ENGLISH} l$

This would express that verbs take DP complements in English, and that in English all heads take their complements to the right.

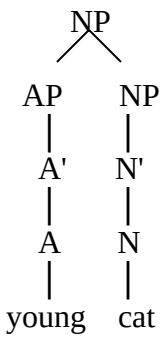
For our purposes we have specified enough of what is universal

Example: Young cat

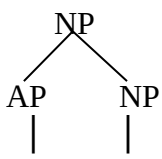
We're interested in English adjectival modification, and in particular in generating *young cat*. In the X-bar perspective given here, what we need to specify in the grammar of English is the following:

1. Lexical item: *young*: category: A
   Lexical item: *cat*: category: N
2. $MOD[N,A] =_{ENGLISH} r$

This is all we need to specify. The grammar will now automatically allow the following tree to be built:



And this tree gets pruned to:



young cat

Syntax-semantics map

Let us now consider the semantics and the syntax/semantics map.
We extend the X-bar theory with a semantics. We make some assumptions about the syntax-semantics map which we would assume to be part of the universal part.

We first consider the basic categories. In the spirit of the minimalist approach to interpretation of type shifting theories, let us assume that the interpretation of adjectives and nouns is as simple as can be. We assume the following universal association:

$A \rightarrow \langle e,t \rangle$
$N \rightarrow \langle e,t \rangle$

Secondly, ignoring co-ordination, let us now look at structure building operations. They are of two kinds: non-branching and binary branching. The universal semantics for non-branching structure is just identity:

$A \rightarrow \alpha'$
$|$
$\alpha \rightarrow \alpha'$

Let us assume that the universal semantics for branching structures is function-argument application:

This means that all three types of branching structures that we have introduced have, at this level of generality, the same interpretation: apply the function to the argument.

What needs to be specified still is what is the function and what is the argument. I will assume here that this too is specified universally, but not universally the same for all binary structures. In particular, I will assume that here the difference between **adjuncts** and **arguments** comes in.
Both the relations COMP(X,Y) and SPEC(X,Y) are typically used for the relation between a role assigner and a role receiver, a function and an argument. I will assume that this is reflected in the syntax/semantics map:

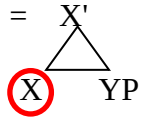
In complementation and specification, the **head** is the semantic function.

I will assume that it is the other way round in modification:

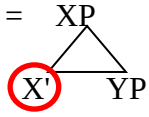
In modification, the **head** is the semantic argument.

This gives us the following universal interpretations:

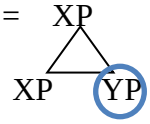
COMP[X,Y] = X' → APPLY[f,a]
f = X f = head



SPEC[X,Y] = XP → APPLY[f,a]
f = X' f = head



MOD[X,Y] = XP → APPLY[f,a]
f = XP f = non-head



Let us assume the following universal specifications

D → <<e,t>,e>
N → <e,t>
V → <e,<e,t>>
I → <<e,t>,<e,t>>

We assume the following particulars about English (which themselves are, of course, instances of a more general pattern):

D, N, V, I ∈ $CAT_{ENGLISH}$

COMP[D,N] $=_{ENGLISH}$ *l*
COMP[V,D] $=_{ENGLISH}$ *l*
COMP[I,V] $=_{ENGLISH}$ *l*
SPEC[I,D] $=_{ENGLISH}$ *r*

And we add the following lexical items, for English, with their interpretation:

Lexical item: the
Category: D
Interpretation: λP.σ(P)

Lexical item: cat
Category: N
Interpretation: CAT

Lexical item: girl
Category: N
Interpretation: GIRL

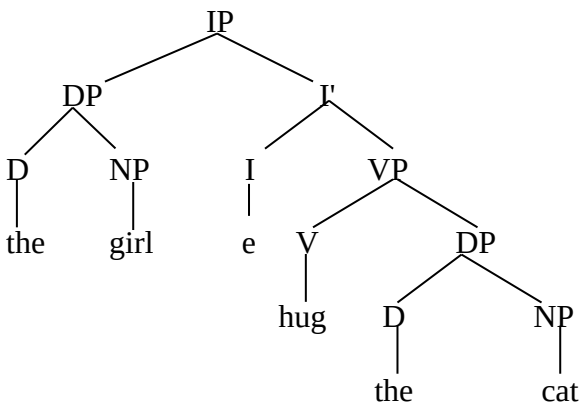
Lexical item: e
Category: I
Interpretation: λP.P
Lexical item: hug
Category: V
Interpretation: HUG

An example: *the girl hugged the cat*

Let us show how, with these assumptions, we can generate sentence (1):

(1) The girl hugged the cat.

With this grammar, we derive the following pruned syntactic tree:



All complements are to the right of their head.
D takes an NP complement.
V takes a DP complement.
I takes a VP complement.
Specifiers are to the left of their head.
I' takes a DP specifier.

Semantic interpretation:
cat → CAT (∈ $CON_{<e,t>}$)
hence: $[_{NP}$ cat] → CAT

the → λP.σ(P)
D is the head of D', the relation is complementation, hence D is the function. Thus we get:
$[_{D'}$ the cat] → APPLY[λP.σ(P),CAT] = σ(CAT)
This is also the interpretation of the DP (inheritance).

hug → HUG (∈ $CON_{<e,<e,t>>}$)
V is the head of V', the relation is complementation, hence V is the function. Thus we get:
$[_{V'}$ hug the cat] → APPLY[HUG,σ(CAT)] = (HUG(σ(CAT)))
This is also the interpretation of the VP.

e → λP.P
I is the head of I', the relation is complementation, hence I is the function. Thus we get:
$[_{I'}$ hug the cat] → APPLY[λP.P, (HUG(σ(CAT)))]
= (HUG(σ(CAT)))

girl→ GIRL (∈ $CON_{<e,t>}$)

hence: [$_{NP}$ girl] → GIRL

the → λP.σ(P)
D is the head of D', the relation is complementation, hence D is the function. Thus we get:
[$_{D'}$ the girl] → APPLY[λP.σ(P),GIRL] = σ(GIRL)
This is also the interpretation of the DP.

I' is the head of IP, the relation is specification, hence I' is the function. Thus we get:
[$_{IP}$ the girl hug the cat] → APPLY[(HUG(σ(CAT))),σ(GIRL)] =
HUG(σ(GIRL),σ(CAT))

We have now set up the grammar in such a way that the language specific semantic assumptions are reduced to a minimum: just the meaning of the lexical items is specified by the language.

But now we have to face the fact that we will get **massive type mismatch**. We see that when we look at the adjective case discussed earlier, and also when we add the universal determiner *every* to the language:

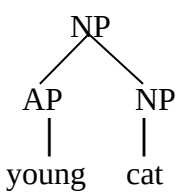
Universal determiners: D → <<e,t>,<<e,t>,t>>

Lexical items:

Lexical item: young
Category: A
Interpretation: YOUNG

Lexical item: every
Category: D
Interpretation: λQλP.∀x[Q(x) → P(x)]

The tree we got for *young cat* was:



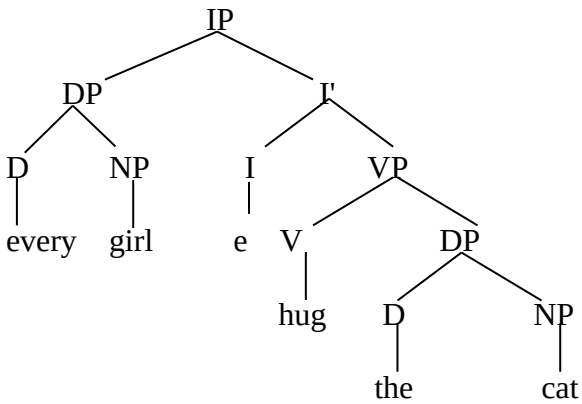
The AP is an NP modifier. In modifier constructions the modifier is the function. Thus the semantic interpretation for this tree is:

APPLY[YOUNG,CAT] where YOUNG, CAT ∈ CON$_{<e,t>}$

We have a type mismatch: the interpretation of the AP should be a function taking the interpretation of the NP as an argument, and it isn't.

Similarly, we get the following syntactic structure for sentence (2):

(2) Every girl hugged the cat.



All complements are to the right of their head.
D takes an NP complement.
V takes a DP complement.
I takes a VP complement.
Specifiers are to the left of their head.
I' takes a DP specifier.

The subject DP gets the following interpretation.
D takes an NP complement, hence the interpretation of D is a function which applies to the interpretation of NP, and we get, as usual, a generalized quantifier:

$[_{DP}$ every girl$] \rightarrow \lambda P.\forall x[GIRL(x) \rightarrow P(x)]$ of type <<e,t>,t>

The interpretation of I' was HUG(σ(CAT)) of type <e,t>.
I' takes a DP specifier, its interpretation is a function on the interpretation of the DP, so we get as the interpretation of the IP:

APPLY[HUG(σ(CAT)), $\lambda P.\forall x[GIRL(x) \rightarrow P(x)]$]

Again, we have a type mismatch: the interpretation of the I' should be a function taking the interpretation of the DP as an argument, and it isn't.

Now we are ready to introduce the idea of type shifting. The idea behind the framework of type driven interpretation is that the semantic interpretation can indeed lead to type mismatch. In the case of type mismatch, the interpretation blocks (or crashes, as we now say), **unless the type mismatch can be resolved**.

What we add to the grammar, in this approach, is a **mechanism for resolving type mismatch**. This mechanism is called a **type shifting theory**, and it consists of a set of **type shifting operations**:

A **type shifting operation** from type a to type b is an operation SHIFT which takes any expression α of type a and maps it onto an expression SHIFT[α] of type b.

A **type shifting theory** is a set of type shifting operations, containing for each type a, the following trivial type shifting operation:

ID:a $\rightarrow$ a

$$ID[\alpha] = \alpha$$

(Allowing identity as a typeshifting rule simplifies the general definitions.)
Resolving type mismatch works as follows.
Before we assumed that:

$$APPLY[\alpha,\beta] = (\alpha(\beta))$$

But that means, in the present context of type mismatch, that we get an unwellformed expression, i.e. something that isn't an expression. Rather than assuming that, we **redefine** the notion APPLY (and similarly the other operations used in the grammar).

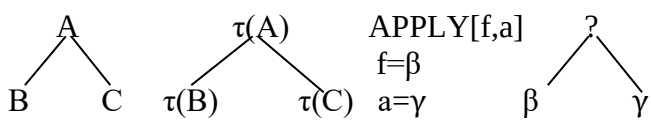
The idea is that the type shifting theory associates with the operation of APPLY and any α and β a set TS[APPLY,α,β,a] of **solutions of the type mismatch of type a**. We define:

$$TS[APPLY,\alpha,\beta,a] =$$

$$\{ ((r(\alpha))((s(\beta)))) \in EXP_a : r,s \in TS\}$$

In general, if the type shifting theory allows more than one resolution for the type mismatch, we will need a theory ranking resolutions and chosing the highest ranking one. In practice, for the purpose of these lecture notes we will only assume typeshifiting theories which are declarative in that the type mismatch can be resolved in only one way, if at all.

So, the idea of type driven interpretation is that you interpret everything as low as you can. In such a theory you may well get to a situation where you have a type mismatch:



Solving the type mismatch is a bit like solving a little jigsaw puzzle.
We have the interpretations of B and C given (derived).
We may well have the *type* of the interpretation of A already given (for independent grammatical reasons).
It may well be given which basic operation must apply here, say functional application (also for independent grammatical reasons),
and it may well be given which is the function and which is the argument, say, specified as above.
Above, we set up our little grammar in such a way that this was given by independent grammatical reasons (but you may not want to follow me in that and change the grammar).
In this context, the type mismatch is resolved if you can find in the type shifting theory a solution for APPLY[β,γ].

We will build the declarative search for a solution into the definition of generalized application APPLY. (This is done to get a procedure that is easy to work with for you!)

Generalized functional application: APPLY

1. If $\alpha \in EXP_{<a,b>}$ and $\beta \in EXP_a$ then:
   $APPLY[\alpha,\beta] = (\alpha(\beta))$

If the condition in (1) doesn't hold then:
2. If $r \in TS$ and $r[\alpha] \in EXP_{<a,b>}$ and $\beta \in EXP_a$ then:
   $APPLY[\alpha,\beta] = (r[\alpha](\beta))$

If the conditions in (1) and (2) do not hold then:
3. If $s \in TS$ and $\alpha \in EXP_{<a,b>}$ and $s[\beta] \in EXP_a$ then:
   $APPLY[\alpha,\beta] = \alpha(s[\beta])$

If the conditions in (1) - (3) do not hold then:
4. if $r,s \in TS$ and $r[\alpha] \in EXP_{<a,b>}$ and $s[\beta] \in EXP_a$
and $r[\alpha](s[\beta])$ is the minimal solution, then:
   $APPLY[\alpha,\beta] = r[\alpha](s[\beta])$

If the conditions in (1)-(4) do not hold then
   $APPLY[\alpha,\beta]$ is undefined.

Let us now give the (starting) type shifting theory. For the moment it consists of four type shifting rules. For notational simplicity, I will collapse the names of the operations, and call all of them: LIFT.

Type shifting: LIFT
Let $P \in VAR_{<e,t>}$, $T \in VAR_{<<e,t>,t>}$, $x,y \in VAR_e$

1. **Argument lift**.
   $LIFT{:}EXP_e \rightarrow EXP_{<<e,t>,t>}$
   $LIFT[\alpha] = \lambda P.P(\alpha)$

2. **Subject lift**.
   $LIFT{:}EXP_{<e,t>} \rightarrow EXP_{<<<e,t>,t>,t>}$
   $LIFT[\alpha] = \lambda T.T(\alpha)$

3. **Object lift**.
   $LIFT{:}EXP_{<e,<e,t>>} \rightarrow EXP_{<<<e,t>,t>,<e,t>>}$
   $LIFT[\alpha] = \lambda T \lambda x.T(\lambda y.[\alpha(y)](x))$

4. **Adjunct lift**.
   $LIFT{:}EXP_{<e,t>} \rightarrow EXP_{<<e,t>,<e,t>>}$
   $LIFT[\alpha] = \lambda P \lambda x.P(x) \wedge \alpha(x)$

[In all cases, we impose a condition that the variables specified do not occur free in α]

The content of the operations (1)-(3) is familiar by now:

Argument lift lifts a proper name α to an expression denoting the set of all properties of α.

Subject lift lifts an intransitive verb to an expression denoting the property which a set of properties X has iff α is one of the properties in that set X.

Object lift lifts a transitive verb to an expression denoting the relation between individuals x and sets of properties X which obtains iff the property of being α-ed by x is one of the properties in that set of properties X.

Finally, **adjunct lift** lifts the predicative adjective to the intersective prenominal interpretation.

We see then that we can now see the operation APPLY as a generalization of functional application.

Functional application, as an operation, requires that the type of the function and the type of the arguments match (<a,b> + a ==> b).

The operation APPLY can apply to a function and an argument even if the types of function and argument do not match,
as long as either the result of lifting the function matches the argument (and you apply the lifted function to the argument) or the function matches the result of lifting the argument (and you apply the function to the lifted argument).

The last case, where you lift both function and argument, you can forget about for the moment, because we will not see an instance of that until much later, when we incorporate Zimmermann's analysis of *seek*.

Since, we have given all our operations the same name, LIFT, (and since it can in our examples always be determined which instance of LIFT is meant), the definition of APPLY becomes:

Generalized functional application: APPLY

1. If $\alpha \in EXP_{<a,b>}$ and $\beta \in EXP_a$ then:
   $APPLY[\alpha,\beta] = (\alpha(\beta))$

If the condition in (1) doesn't hold then:
2. If $LIFT[\alpha] \in EXP_{<a,b>}$ and $\beta \in EXP_a$ then:
   $APPLY[\alpha,\beta] = (LIFT[\alpha](\beta))$

If the conditions in (1) and (2) do not hold then:
3. If $\alpha \in EXP_{<a,b>}$ and $LIFT[\beta] \in EXP_a$ then:
   $APPLY[\alpha,\beta] = \alpha(LIFT[\beta])$

If the conditions in (1) - (3) do not hold then:
4. if $LIFT[\alpha] \in EXP_{<a,b>}$ and $LIFT[\beta] \in EXP_a$
and $LIFT[\alpha](LIFT[\beta])$ is the minimal solution, then:
   $APPLY[\alpha,\beta] = LIFT[\alpha](LIFT[\beta])$

If the conditions in (1)-(4) do not hold then
   $APPLY[\alpha,\beta]$ is undefined.

Example: young cat

Let us come back to the example of the X-bar based grammar.
The only language specific assumptions we made were the specification of the lexical items, and some decisions about the left-right order of heads. We generated a syntactic structure for *young boy*. But the derivation got stuck because we got a type mismatch in:

$APPLY[YOUNG,CAT]$ where $YOUNG, CAT \in CON_{<e,t>}$

The type shifting theory LIFT and the new definition of APPLY, now tells us that this type mismatch is resolved as follows:

$APPLY[YOUNG,CAT] =$
$LIFT[YOUNG](CAT) =$
$\lambda P \lambda x.P(x) \wedge YOUNG(x)\ (CAT) =$
$\lambda x.CAT(x) \wedge YOUNG(x)$

Similarly, in the derivation of sentence (2):

(2) Every girl hugged the cat

we got stuck, because at the level of the interpretation of the IP we got the type mismatch:

APPLY[HUG(σGIRL), λP.∀x[GIRL(x) → P(x)]]

Again, this type mismatch now gets resolved as follows:

APPLY[HUG(σCAT), λP.∀x[GIRL(x) → P(x)]] =
LIFT[HUG(σCAT)] (λP.∀x[GIRL(x) → P(x)]) =
λT.T(HUG(σCAT)) (λP.∀x[GIRL(x) → P(x)]) =
λP.∀x[GIRL(x) → P(x)] (HUG(σCAT)) =
∀x[GIRL(x) → HUG(σCAT) (x)] =
∀x[GIRL(x) → HUG(x,σCAT)]

3.4. A little grammar

In this fragment we give the basic analysis of noun phrases, intransitive verb phrases, (extensional) transitive verb phrases and the transitive verb *be*. For *be* we will at first follow Montague's analysis, and change that in a later chapter.

The **categories** of our fragment are D,N,V,I:

$D,N,V,I \in CAT$

The type assignments to the categories are as follows:

$D \rightarrow e, <<e,t>,e>, <<e,t>,<<e,t>,t>>$
$N \rightarrow <e,t>$
$V \rightarrow <e,t>, <e,<e,t>>$
$I \rightarrow <<e,t>,<e,t>>$

Complements and specifiers are just as before. We specify:

$<D,N>,<V,D>,<I,V> \in COMP_{ENGLISH}$
$<I,D> \in SPEC_{ENGLISH}$

And we specify:

for all $<X,Y> \in COMP_E$: $COMP[X,Y] =_E l$
for all $<X,Y> \in SPEC_E$: $SPEC[X,Y] =_E r$

The grammar contains the following lexical items:

Let RONYA, ANNA $\in CON_e$, CAT, GIRL, PURR $\in CON_{<e,t>}$,
HUG $\in CON_{<e,<e,t>>}$ x,y $\in VAR_e$, P,Q $\in VAR_{<e,t>}$.

Lexical item: ronya
Category: D
Interpretation: RONYA

Lexical item: anna
Category: D
Interpretation: ANNA

Lexical item: the
Category: D
Interpretation: $\lambda P.\sigma(P)$

Lexical item: every

Category: D
Interpretation: $\lambda Q \lambda P . \forall x [Q(x) \rightarrow P(x)]$

Lexical item: a
Category: D
Interpretation: $\lambda Q \lambda P . \exists x [Q(x) \wedge P(x)]$

Lexical item: no
Category: D
Interpretation: $\lambda Q \lambda P . \neg \exists x [Q(x) \wedge P(x)]$

Lexical item: cat
Category: N
Interpretation: CAT

Lexical item: girl
Category: N
Interpretation: GIRL

Lexical item: purr
Category: V
Interpretation: PURR

Lexical item: hug
Category: V
Interpretation: HUG

Lexical item: be
Category: V
Interpretation: $\lambda y \lambda x . x = y$

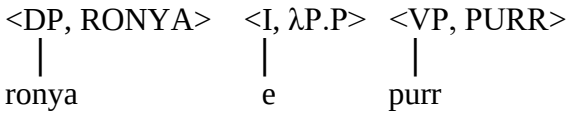
Lexical item: e
Category: I
Interpretation: $\lambda P . P$

The semantic interpretation is specified as before. In non-branching cases we get identity, in branching cases APPLY[f,a], where f is the head in complementation and specification, and the non-head in modification.

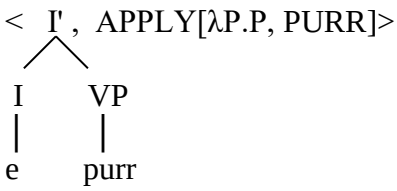
EXAMPLES
In all the following examples I will take alphabetic variants where useful, without mentioning it.

(1) Ronya purrs

From the lexical items *ronya*, e and *purr* we generate:

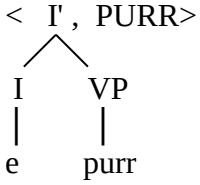


The I takes the VP as a complement, which gives:

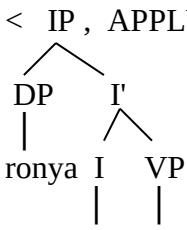


APPLY[λP.P, PURR]
= λP.P[PURR] [def. APPLY]
= PURR [λ-conversion]

So we get:



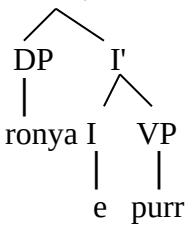
The I' takes the DP as a specifier, which gives:



e purr

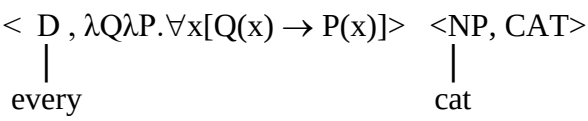
APPLY[PURR, RONYA]
= PURR(RONYA) [def. APPLY]

So we get:
< IP , PURR(RONYA)>

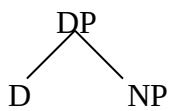


(2) Every cat purrs.

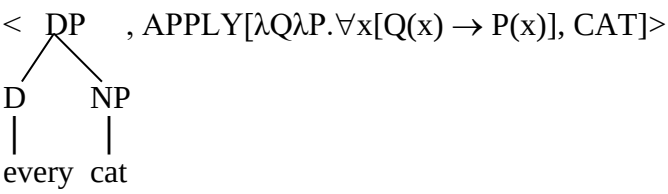
From the lexical items *every* and *cat* we get:



The determiner takes the NP as a complement:

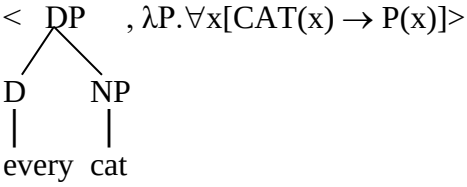


So we get:

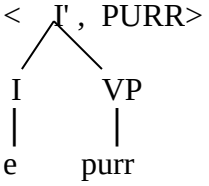


APPLY[λQλP.∀x[Q(x) → P(x)], CAT]
= λQλP.∀x[Q(x) → P(x)] (CAT) [def. APPLY]
= λP.∀x[CAT(x) → P(x)] [λ-con.]

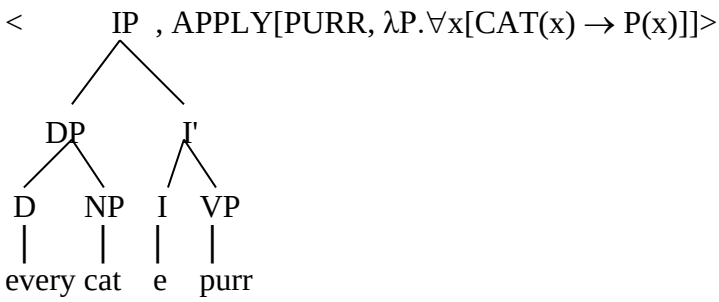
So we get:



This DP can be the specifier of the following I':

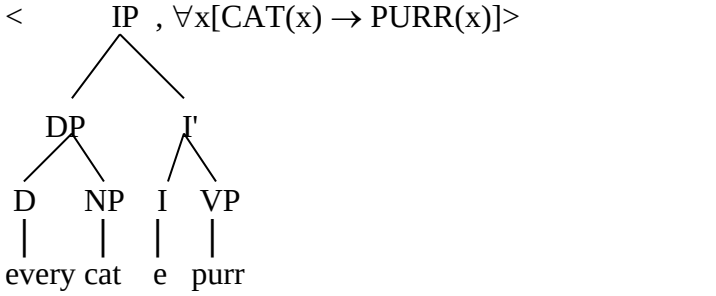


So we get:



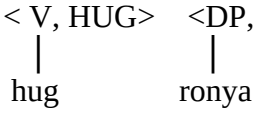
APPLY[PURR, λP.∀x[CAT(x) → P(x)]]
= LIFT[PURR] (λP.∀x[CAT(x) → P(x)]) [def. APPLY]
= λT.T(PURR) (λP.∀x[CAT(x) → P(x)]) [def. LIFT]
= λP.∀x[CAT(x) → P(x)] (PURR) [λ-con.]
∀x[CAT(x) → PURR(x)] [λ-con.]

So we get:

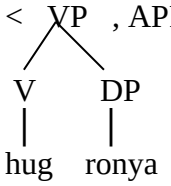


(3) Anna hugs Ronya.

from the lexical items *hug* and *ronya* we get:

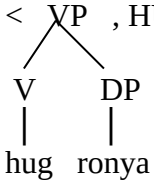


The transitive verb can take the DP as a complement, V' and VP get identified, and we get:

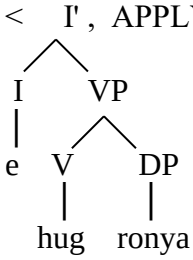


APPLY[HUG, RONYA]
= HUG(RONYA) [def. APPLY]

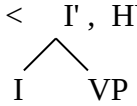
So we get:

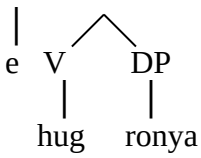


I takes the VP as a complement:



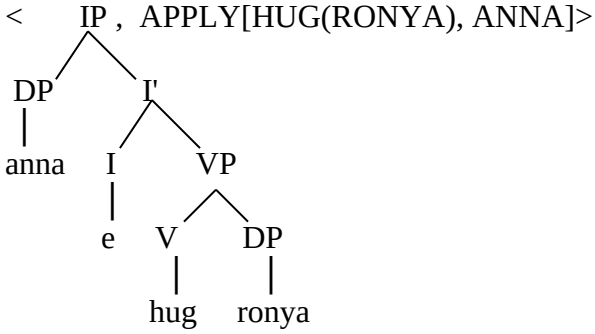
which is:





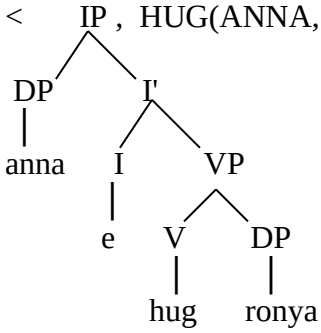
This I' can take as a specifier:

which gives:



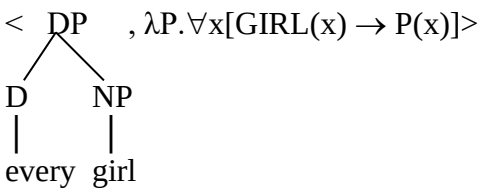
APPLY[HUG(RONYA), ANNA]
= [HUG(RONYA)](ANNA) [def. APPLY]
= HUG(ANNA, RONYA) [rel. notation]

So we get:

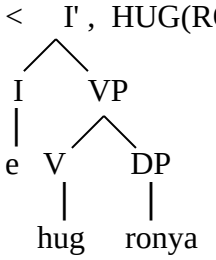


(4) Every girl hugs Ronya

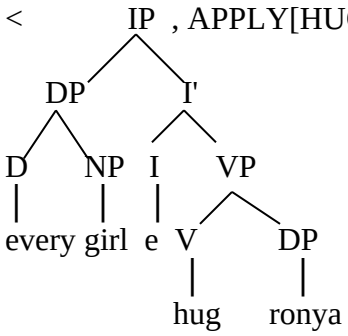
Above we have given the derivation of the DP *every cat*, so similarly we get *every girl*:



Also above we have given the derivation of the I' *hug Ronya*:

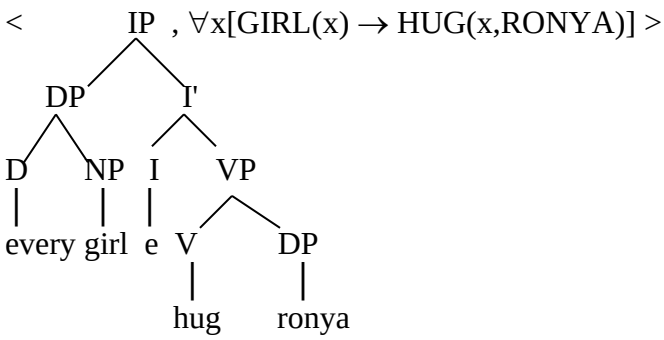


This I' can take this DP as a specifier, and we get:



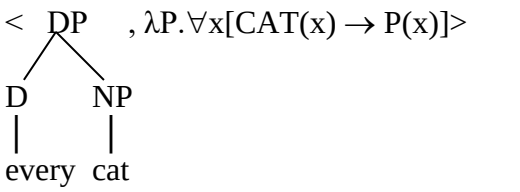
APPLY[HUG(RONYA), λP.∀x[GIRL(x) → P(x)]
= LIFT[HUG(RONYA)] (λP.∀x[GIRL(x) → P(x)]) [def. APPLY]
= λT.T(HUG(RONYA)) (λP.∀x[GIRL(x) → P(x)]) [def. LIFT]
= λP.∀x[GIRL(x) → P(x)] (HUG(RONYA)) [λ-con]
= ∀x[GIRL(x) → [HUG(RONYA)](x)] [λ-con]
= ∀x[GIRL(x) → HUG(x, RONYA)] [rel. notation]

So we get:

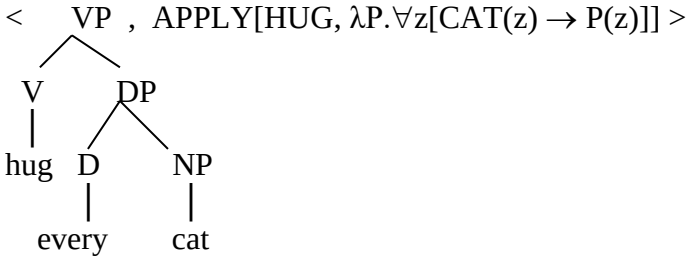


(5) Anna hugs every cat.

The derivation of the DP *every cat* was given above:

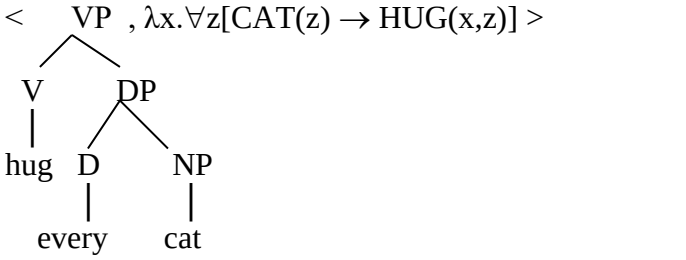


The V *hug* takes this as a complement, and we get:



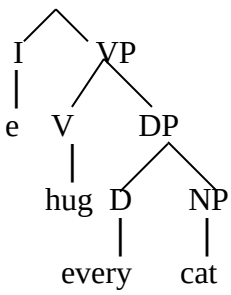
APPLY[HUG, λP.∀z[CAT(z) → P(z)]]	
= LIFT[HUG] (λP.∀z[CAT(z) → P(z)])	[def APPLY]
= λTλx.T(λy.[HUG(y)](x)) (λP.∀z[CAT(z) → P(z)])	[def LIFT]
= λx.[λP.∀z[CAT(z) → P(z)]](λy.[HUG(y)](x))	[λ-con]
= λx.∀z[CAT(z) → [λy.[HUG(y)](x)](z)]	[λ-con]
= λx.∀z[CAT(z) → [HUG(z)](x)]	[λ-con]
= λx.∀z[CAT(z) → HUG(x,z)]	[rel not]

So we get:



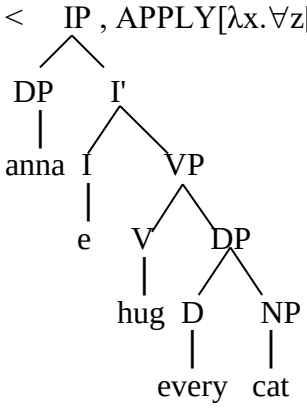
I takes this VP as a complement, the semantics is, as usual identity, and we get:

< I' , λx.∀z[CAT(z) → HUG(x,z)] >



This I' can take the DP *anna* as a specifier, which gives:

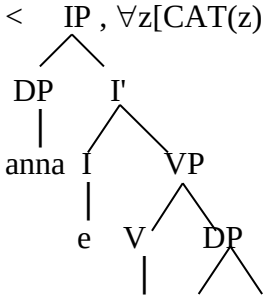
< IP , APPLY[λx.∀z[CAT(z) → HUG(x,z)], ANNA] >

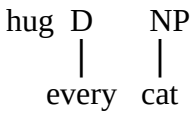


APPLY[λx.∀z[CAT(z) → HUG(x,z)], ANNA]
= λx.∀z[CAT(z) → HUG(x,z)] (ANNA) [def APPLY]
= ∀z[CAT(z) → HUG(ANNA, z)] [λ-con]

So we get:

< IP , ∀z[CAT(z) → HUG(ANNA, z)] >

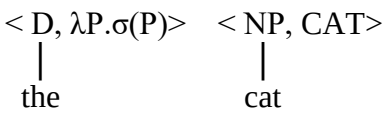




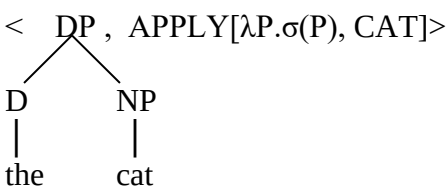
We will now discuss some examples which show the essence of Montague's analysis of the verb *be*.

(7) Ronya is the cat.

From the lexical items *the* and *cat*, we get:

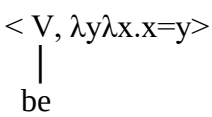


And this gives the DP:

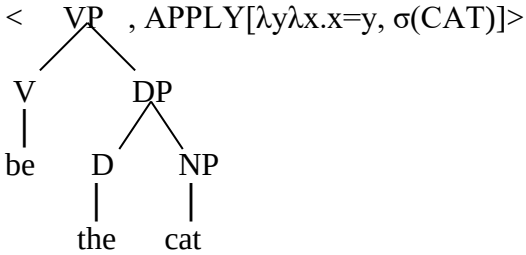


APPLY[λP.σ(P), CAT]
= λP.σ(P)(CAT) [by def. APPLY]
= σ(CAT) [by λ. conversion]

The lexical item for *be* gives us:

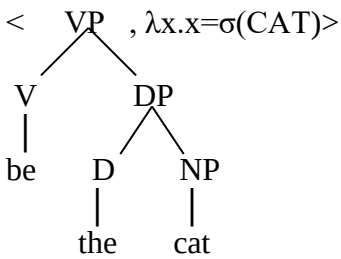


The transitive verb *be* takes the DP as a complement, and we get:

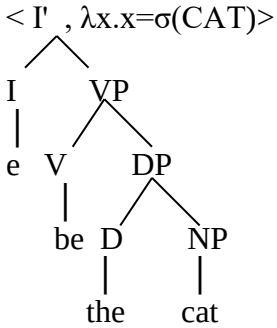


APPLY[λyλx.x=y, σ(CAT)]
= λyλx.x=y (σ(CAT)) [def APPLY]
= λx.x=σ(CAT) [λ-con]

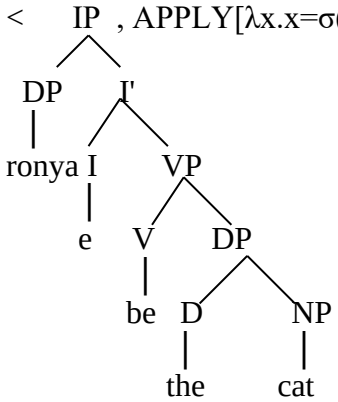
So we get:



This gives the I':

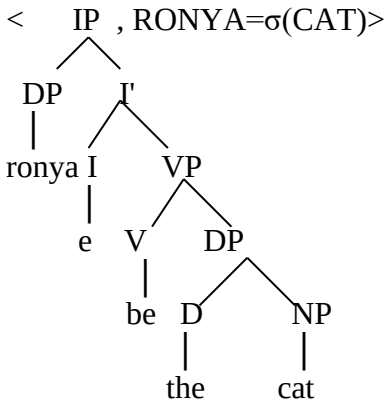


This I' takes the DP *ronya* as a specifier, and we get:



APPLY[λx.x=σ(CAT), RONYA]
= λx.x=σ(CAT)(RONYA) [def APPLY]
= RONYA=σ(CAT) [λ-con]

So we get:


< IP , RONYA=σ(CAT)>
DP
I'
ronya
I
VP
e
V
DP
be
D
NP
the
cat

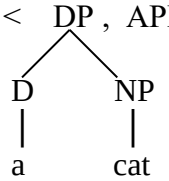
(8) Ronya is a cat.

From the lexical item *a* we get:

< D, λQλP.∃x[Q(x) ∧ P(x)]>
|
a

This takes the NP *cat* as a complement, and we get the DP:

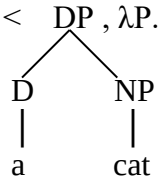
< DP , APPLY[λQλP.∃z[Q(z) ∧ P(z)], CAT]>



APPLY[λQλP.∃z[Q(z) ∧ P(z)], CAT] [def. APPLY]
= λQλP.∃z[Q(z) ∧ P(z)](CAT) [λ-con.]
=λP.∃z[CAT(z) ∧ P(z]

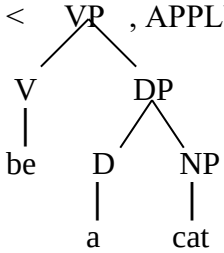
So we get:

< DP , λP.∃z[CAT(z) ∧ P(z)]>



The transitive verb *be* takes this as a complement, and we get:

< VP , APPLY[λyλx.x=y, λP.∃z[CAT(z) ∧ P(z)]>



APPLY[λvλu.u=v, λP.∃z[CAT(z) ∧ P(z)]]

= LIFT[λvλu.u=v] (λP.∃z[CAT(z) ∧ P(z)]) [def APPLY]

= λTλx.T(λy.[[λvλu.u=v](y)](x)) (λP.∃z[CAT(z) ∧ P(z)]) [def LIFT]

= λTλx.T(λy.[λu.u=y](x)) (λP.∃z[CAT(z) ∧ P(z)]) [λ-con]

= λTλx.T(λy.x=y) (λP.∃z[CAT(z) ∧ P(z)]) [λ-con]

= λx.[λP.∃z[CAT(z) ∧ P(z)]](λy.x=y) [λ-con]

= λx.∃z[CAT(z) ∧ [λy.x=y](z)] [λ-con]

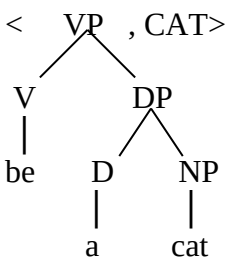
= λx.∃z[CAT(z) ∧ x=z] [λ-con]

= CAT [see below]

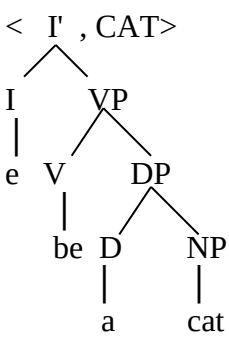
The last equivalence may require some comment:
This reduction holds because of the following obvious classical logical tautologies:
-If you're a cat, then there is something that is a cat and that is identical to you.
-If there is something that is a cat and that is identical to you, then you are a cat.
Given this:

λx.∃z[CAT(z) ∧ x=z] = CAT [extensionality]

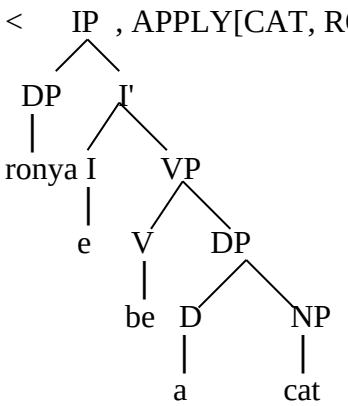
So we get:



We get the I':

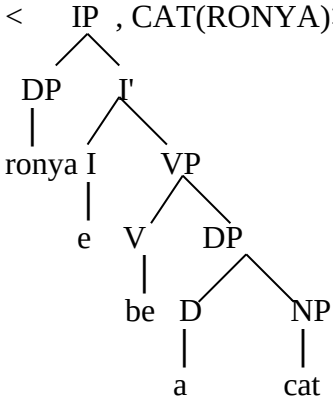


This takes the DP *ronya* as a specifier, and we get:



APPLY[CAT, RONYA]
= CAT(RONYA) [def APPLY]

So we get:



We see that while *be* is interpreted unambiguously as the identity relation, we derive nevertheless an identity meaning in (7), but a predicative meaning in (8). Moreover, we get a inclusive reading in (9):

(9) Every whale is a mammal.

be a mammal gets interpretation MAMMAL.

Hence, *every whale is a mammal* will be derived in the same way from here as *every cat purrs* and it will be derived with meaning:

$\forall x[\text{WHALE}(x) \rightarrow \text{MAMMAL}(x)]$

Thus whereas earlier philosophical literature (e.g. Hintikka) claimed that *be* was ambiguous, even threeway ambiguous,
Hintikka assumed a *be* of identity, a *be* of predication, and a *be* of subset:

(10 Whales are mammals $\text{WHALE} \subseteq \text{MAMMAL}$

Montague gave a unified analysis for all three readings: identity, predication., subset.
He didn't talk about the subset reading, but if we assume that the generic in (10) is similar to (9), then it is not reasonable to encode the subset properties of the generic in the meaning of *be* rather than in an implicit generic operation.

One problem with Montague's analysis is that it is too general.
The grammar produces unproblematically sentence (11) with meaning (11'):

(11) Anna is every professor.
(11') $\forall x[\text{PROFESSOR}(x) \rightarrow x=\text{ANNA}]$

The problem is that (11) is not a felicitous sentence.
Now, of course, in every situation where there is more than one professor, (11') is systematically false. So maybe that is why it is infelicitous.
The problem is that also in a situation where there is exactly one professor, Anna, uttering (11) is infelicitous, but (11') is true.
This problem is discussed in Partee 1987. We will discuss it further in a later chapter.

As Partee argues, the problem is part of a more general problem with the analysis:
be is not a transitive verb, but a copula (an I element).
And it isn't clear that we want to interpret *be* as identity, because we get the same interpretations in languages where the copula is null,
and assigning real lexical content to a null copula is something we want to avoid.

These problems are addressed in a special part of the type shifting theory call the Partee Triangle, and we will discuss this later.

3.5 Typeshifting theories.

I would like to present you with a general theory of typeshifting principles. But I don't have such a theory. Not that such theories don't exist. A lot of work has been done in categorial grammar on developing a logical perspective on typeshifting theories.
The basics of this you can see in the principle of functional application:

α	β	(α(β))
<a,b>	+ a	→ b

This type combination principle looks much like the logical principle of *modus ponens*, and indeed the interpretation of type combination as a *type inference principle* forms the basis of the general logical perspective of categorial grammar:

> a grammatical <a,b>-interpretation for α and a grammatical a-interpretation for β, derive a grammatical b-interpretation for the complex.

Type shifting principles similarly are interpreted as principles *deriving* grammatical interpretations on different types.

α	λP.P(α)
a	→ <a,<a,t>>

a grammatical a-interpretation for α, derives a grammatical <a,<a,t>> interpretation for α as well.

In this perspective, you can then start to formulate logical grammars as systems of axiomas (like the above type lifting rule) and derivation rules (like functional application, interpreted as Modus Ponens on types.). In this way, logical axiom system forms the basis of *deriving* complex type shifting rules, determining a grammar in a systematic way.

Very beautiful type shifting theories have been developed and studied in this framework.

However, it must also be admitted that, in relation to linguistic puzzles, the puzzles that interest linguists most, such beautiful theories quite generally overgenerate considerably (i.e. derive wrong structures with the right meanings, or right structures with wrong meanings, or just wrong structures with wrong meanings, besides the beauties: right structures with right meanings).
And the process of curtailing the overgeneralization leads to theories that are rather less elegant and a lot more stipulative.

I myself tend to treat typeshifting rules more on a one-to-one basis.
In the context of a particular linguistic puzzle, I try to show that a particularly elegant and simple and cross-linguistically relevant type shifting operation can be used to solve my problem. That counts, I would then argue, as linguistic support for that particular rule.

So, it's not that I am against a general theory. I just don't have one, and I am more concerned with trying to solve the problems, than with formulating a general architecture of type shifting.
Note that I *don't* regard this as a virtue, something to be proud of, and I *do* think we need more general thinking about type shifting theories, but it does seem to me that the general theories I am familiar with get away from the linguistic details too easily.

So, I don't have a theory, but at least I have a sort of field guide: how to recognize a type shifting principle in the wild.

Type shifting principles, as practiced in the field, come in four types.

1. Homomorphic typelifting.

This is, in essence, what Montague gave us, and what we have been developing here:
We use type shifting principles to define functions at a higher (resolution) type that do at the higher type what the corresponding lower function does at the lower type.

2. Domain shifts.

There are many linguistic domains that are naturally paired, and that we assume are linked via pairs of operators bringing you from the one to the other. These operators are often regarded as type shifting or sort shifting operators.

-Mass-count
Coffee is a mass noun, *the coffee in the cup* denotes a mass object, stuff:
✓*much coffee/#many coffee*
✓*much of the coffee in the cup/#many of the coffee in the cup*.

Both, like *each* is a marker of *count* DPs: *both* is *each* on a domain of two; *each* in *the boys each sang a song* distributes the predicate *sang a song* to the individual boys. The 'individual' parts of mass objects, if there are such, are not similarly available for the distributitive operators *each* and *both*:

(1). a. ✓The *boys* were *both* blond.
b. #The *mud* was *both* brown/#The *muds* were *both* brown.
c. ✓*Both boys* were bold/✓*Both* of the *boys* were blond.
d. #*Both mud(s)* were brown/#*Both* of the *mud(s)* were brown.

But the mass object *the coffee in the cup* can **shift** to a count object as in (2b):

(2) a. The coffee in the cup and the coffee in the pot weighed 150 grams.
b. ***Both*** the coffee in the cup **and** the coffee in the pot contained strychnine.

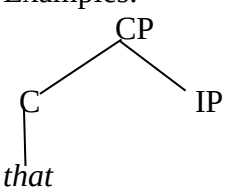
As (2a) shows, we are grammatically able to regard the interpretation of *the coffee in the cup and the coffee in the pot* as a single mass object, an object that weighs 150 grams.

But, as (2b) shows, we are *also* able to regard the interpretation of this expression as a conjunction of two **count** objects, since *both* distributes the predicate *contained Strychnine* to those two.
This means that we can **shift** the interpretation of *the coffee in the pot* from an interpretation as a mass entity (stuff) to an interpretation as a count entity (a single object).
This shift is called **packaging**: treat a non-countable mass entity as a single countable count entity.
The inverse shift is called **grinding**, and you find it in English for count nouns in contexts that do not allow bare singular count nouns. Many of the more gruesome examples are due to Geoffrey Pelletier, who has studied the phenomenon extensively (the second example is from Rothstein 20??):

(3) a. After the accident with the fan, there was *chiwawa* all over the wall.
b. After the failed repair attempt, there was *watch* all over the table.

-Intensional-extensional: Individual concepts. I will only do intensionality in the next chapter, but we discussed this in Foundations and gave a first introduction to the operations of intensionalization ^ and extensionalization $^{\vee}$.
These operations are generally taken to be type shifting operations.

Examples:



In intensional contexts we want the type of the complement to be a proposition, which, already in Foundations we assumed to be a function from worlds to truth values (a set of worlds). Expressions denoting propositions are going to be of type <s,t> in the next chapter.
In the above case, we have a situation in which the IP is of type t and the CP is of type <s,t>.
This suggests that C should be of type <t,<s,t>>.
However, we will show in the next chapter that for solid logical reasons this cannot be the right type. What we will argue is that the type of C is <<s,t>,<s,t>>, a function from propositions to propositions and, in fact, that the meaning of *that* is trivial:

that → λp.p with p ∈ $VAR_{<s,t>}$, the identity function.

This is good, because *that* often is optional here.
But that means that there is a type mismatch: APPLY[λp.p, φ], where φ ∈ $EXP_{t.}$
This type mismatch is resolved by taking intensionalization to be a type shifting operation:

INT: a → <s,a>
INT[α] = ^α More details in the next chapter.

Now APPLY[λp.p, φ] resolves as λp.p(INT[φ]) = λp.p(^φ) = ^φ

The earlier discussion of Montague's strategy suggests that we want to allow in certain cases

composition of type shifting rules to count as type shifting rules in their own right.

Thus, we have LIFT:e → <<e,t>,t> with LIFT[α] = λP.P(α)
But if we assume, with Montague, that the type of SEEK is <**<s,<<e,t>,t>>,** <e,t>>, then APPLY[SEEK, RONYA] requires lifting RONYA not just to <<e,t>,t>, but to <s,<<e,t>,t>)>
The function that does that is:

INT ∘ LIFT: e → <s,<<e,t>,t>>
INT ∘ LIFT[α] = ^λP.P(α) apply intensionalization after lifting to <<e,t>,t>

You see here the temptation of a logical theory of type shifting rules: All such theories that I am familiar with, derive the composition of two type shifting rules as a type shifting rule from the type shifting logic.

-sums and groups - collective/distributive interpretations
Such shifts have been proposed in Landman 1989 for dealing with the distinction between collective and distributive readings of plural noun phrases.

(5) The boys met in the park and took off their clothes to swim.

Met in the park is a property of the boys as a group, while *took off their clothes to swim* is a property that distributes to the individual boys. In the theory of Landman 1989, you can shift between *the boys as a singular group* and *the boys as a plurality, a sum*, where the latter associates with distributivity.
Here too there is a parallel with the mass-count case above:

(6) The boys and the girls separated and met in different rooms

Here it is useful to regard the expression *the boys and the girls* as a sum of two groups, the group of boys and the group of girls. This is a *gridded* interpretation, where distribution is not to the individuals making up the totality of boys and girls, but to objects at an intermediate level of grid.

-Kinds and stages of kinds – generic and episodic properties (Carlson 1977)

(7) Beavers *build dams* and *kept me awake yesterday with their noise*.

Carlson argues that the bare plural *beavers* in generic sentence (7) patterns on a battery of tests with proper names and hence should be analyzed as names of entities, kinds.
He argues that there are two kinds of predicates relating to kinds: *individual level predicates* like the generic predicate *build dams*, which are properties of the kind, and *stage level predicates* or *episodic predicates* which relate the kind to its instances, which Carlson calls stages of the kind: *kept me awake yesterday* is a predicate of individuals, but in the conjunction is predicated of *beavers*, which denotes a kind.
Type shifting allows us to derive a predicate of kinds:

The property that a kind has if some of its instances kept me awake last night.

Notice that the semantic analysis of the generic is much more complex. When we state that beavers build dams, there is of course a relation between dam building and the instances of

the kind, but that is not a straightforward relation.

> *Dogs* have four legs ~ All [non-exceptional] dogs have four legs.
> *Frenchmen* eat horsemeat ~ Some [not so exceptional] frenchmen eat horsemeat
> Mary deals with *letters from Antarctica* ~ If ever there are letters from Antarctica, which there have never been, Mary would deals with them.

This is different from episodic readings, which are just existential instantiations.

3. The Partee triangle

Partee discusses the set of type shifting principles potentially involved in relating the argument types e and <<e,t>,t> to each other and to the predicate type <e,t>.
More discussion in a later chapter.

Note that the shifts involved in domain-shift are quite often not interpreted as principles to resolve mismatches, but more as free principles that allow, in the appropriate contexts, shift freely. But see Rothstein 2011 (Riga paper) for arguments that grinding in English be regarded as a mismatch resolution principle.

4. Grammatical Relations (in the sense of Dowty 1982).

These are in many ways among the most interesting candidates for type shifting principles (in particular in the context of type driven resolution).
Here we are concerned with:
-Operations that in essence come up in linguistic contexts again and again.
-Operations that are usually natural operations from a conceptual mathematical point of view.
-Operations that arguably are operative in more than one linguistic domain (like both in the verbal domain and in the nominal domain). In fact, operations that can be lexicalized in one domain, but not in the other domain, but are semantically available in the other domain as well.
-Operations that are arguably cross-linguistically relevant, and again, that are *not* lexicalized in many languages, but that *are* lexicalized in others.

The most obvious examples are the interpretations of indefinite and of definite articles:

> -Existential closure: ∃
> -Definiteness: σ

Thus, for example many languages do not mark (in)definiteness or mark one of them, though arguably, the semantic operations are operative. English marks definiteness in the determiner system, but, as Polly Jacobson has argued, a natural analysis of English free relatives (*whoever stole my watch will be found/whoever returns my watch, gets a reward*) is to assume that they denote predicates of type <e,t>, and involve shifting to type e with type shifting operation σ. Thus, in English relative clauses, σ would be available as a typeshifting

operation. It has been argued that this is the natural case in, say, Japanese.

In many domains, existential closure is the natural operation of **projection**.

Thus, passive can be regarded as relating to the operation of **converse** in (8a) and of **first projection** in (8b):

(8) a. Mary was kissed by Jane $(KISS^{C}(JANE_{[by]}))(MARY))$
b. Mary was kissed $\lambda x.\exists y[KISS(x,y]\ (MARY)$

(Landman 2004 uses conversion as a typeshifting operation in the context of the analysis of definiteness effects.)

Identifying two arguments of a relation is another operation that is realized in different ways in the grammar, via reflexive pronouns, clitics, lexical semantics, etc.

reflexive: $\alpha \quad \rightarrow \quad \lambda x.\alpha(x,x)$

In fact, conjunction itself is a natural principle for a type shifting principle. Look at predicate conjunction:

$\lambda Q\lambda P\lambda x.P(x) \wedge Q(x)$

To treat this as a type shifting principle, we only need to think of this as a function on the first conjunct:

$\alpha \quad \rightarrow \quad \lambda P\lambda x.P(x) \wedge \alpha(x)$

But this means that we naturally derive the adjunction typeshifting operation as an instance of conjunction.

Another natural operation: in the theory of Link 1982, semantic plurality is the natural operation of closure under sum (see below), and this too has been argued (eg. in Landman 2000) to be a type shifting operation for predicates that do not mark semantic plurality, namely in the verbal domain.

Similarly, many aspectual operators, low ones, shifting between verb classes, and higher ones, like the perfect and the progressive have been studied as potentially type shifting operations.

In sum, what Dowty 1982 called *grammatical relations* are the natural domain to look for typeshifting principles.