September 19, 2014

1 Some formal bits

1.1 Type theory

An expression's **type** tells you a number of things:

1. Whether the expression denotes a function.
2. If so, how many arguments that function takes...
3. And what sorts of arguments they are.

Here are some examples. For instance, a transitive verb like met, type $\langle e, \langle e, t\rangle\rangle$, denotes a two-place function. It takes an individual, then another individual, and finally returns a truth value.

Type	Alternatively	Things of that type
e	e	John, Mary, Uni
t	t	John left, Uni licked Puffy
$\langle e, t\rangle$	$e \to t$	left, licked Puffy, boy, out, in New Jersey
$\langle e, \langle e, t\rangle\rangle$	$e \to e \to t$	licked, in, part, showed Puffy
$\langle e, \langle e, \langle e, t\rangle\rangle\rangle$	$e \to e \to e \to t$	showed, gave
$\langle t, t\rangle$	$t \to t$	it isn't the case that
$\langle e, e\rangle$	$e \to e$	of
$\langle \langle e, t\rangle, \langle e, t\rangle\rangle$	$(e \to t) \to e \to t$	is, a
$\langle t, \langle t, t\rangle\rangle$	$t \to t \to t$	and, or

More formally, the **set of all types** is the smallest set T such that:

1. $e \in T$.
2. $t \in T$.
3. If $\sigma \in T$ and $\tau \in T$, $\langle \sigma, \tau\rangle \in T$ (alternatively, $\sigma \to \tau \in T$).

There are a number of types in T that we haven't dealt with yet (in fact, infinitely many). Today we will see expressions with type $\langle \langle e, t\rangle, e\rangle$ (things that denote functions from properties to individuals). Next week we will see expressions with type $\langle \langle e, t\rangle, t\rangle$ (functions from properties to truth values).

1.2 Lambda notation

Instead of writing the following...

$$f : \text{for all } x,\ f(x) = \Phi$$

We will write the following:

$$\lambda x.\Phi$$

Lambda terms like this are just an alternative way to write down functions. We'll use this notation cause it's concise, standardized, and more regimented than our somewhat informal conventions so far.

Example. We will use the convention of naming the denotation of words with italics and a prime:

$$[\![\text{meows}]\!] = \lambda x.\, x \in \{y : y \text{ meows}\} = \lambda x.meows'(x)$$

Applying a λ-term to its argument works no differently from how we've been doing it. You replace a variable (say, x) with the argument:

$$(\lambda x.\Phi)(a) = \Phi[a/x]$$

$\Phi[a/x]$ is the formula just like Φ, but with all the **free occurrences** of x in Φ replaced by a. This simplification is known as a β-equivalence or β-reduction.

Informally, x is free (equivalently, not bound) in Φ iff there's no λx scoping over x in Φ (λx scopes over x in $\lambda x.f(x)$ but not in $(\lambda x.f(y))(x)$):

$$\overbrace{\lambda x.\underbrace{f(x)}_{x \text{ is free}}}^{x \text{ is bound}}$$

How about the following? Is the first x free in Φ? Is the second? What does this tell you about how to apply this function to some a?

$$\lambda x.\underbrace{f(\overset{1}{x})(\lambda x.g(\overset{2}{x}))}_{\Phi}$$

Some other important equivalences:

1. α-equivalence: $\lambda x.\Phi = \lambda y.\Phi[y/x]$
   $\lambda x.meows'(x) =_\alpha \lambda y.meows'(y)$

2. η-equivalence: $f = \lambda x.\, f(x)$
   $\mathit{licked}' =_\eta \lambda y.\mathit{licked}'(y) =_\eta \lambda y.\lambda x.\mathit{licked}'(y)(x)$

Be very careful about variables. β-conversion cannot turn a free variable into a bound one, as in the first example below. Whenever there is the potential for **variable capture**, rename bound variables using α-equivalence:

1. $(\lambda x.\, \lambda y.\, f(x)(y))(y) \neq \lambda y.\, f(y)(y)$
2. $(\lambda x.\, \lambda z.\, f(x)(z))(y) = \lambda z.\, f(y)(z)$

We will use subscripts on arguments to talk about domain of a function. For example, $\lambda x_e.\Phi$ is a function of type $\langle e, \sigma\rangle$, for some σ.

Note on notation:

1. We'll write formulae like $\lambda x_\sigma.\lambda y_\tau.\Phi$
2. H&K write formulae like $\lambda x : x \in D_\sigma.[\lambda y : y \in D_\tau.\Phi]$

This may seem like a bewildering number of conventions to master. But once you get the hang of it, it's incredibly natural. We will have a bunch of practice with the notation.

2 Modification

2.1 Complex predicates

Examples of complex predicates:

(1) Uni is a white cat.

(2) A dilapidated house near New Brunswick collapsed.

(3) Kaline is a grey cat in Texas fond of Joe.

A natural thing to suppose is an alternative mode of combination called **predicate modification**. Combines two properties by intersecting them:

> **Predicate modification**
> If A is a branching node whose daughters B and C are both of type $\langle e, t\rangle$, $[\![A]\!] := \lambda x.\, [\![B]\!](x) = [\![C]\!](x) = 1$.

> **Functional application**
> If A is a branching node with a daughter B of type β and a daughter C of type $\langle \beta, \alpha\rangle$, $[\![A]\!] := [\![C]\!]([\![B]\!])$.

Aside from our lexical meanings, **everything** we've had to say about interpretation is summed up in these two rules.

2.2 Example

Example analysis of a grey cat in Texas fond of Joe in Figure 1.

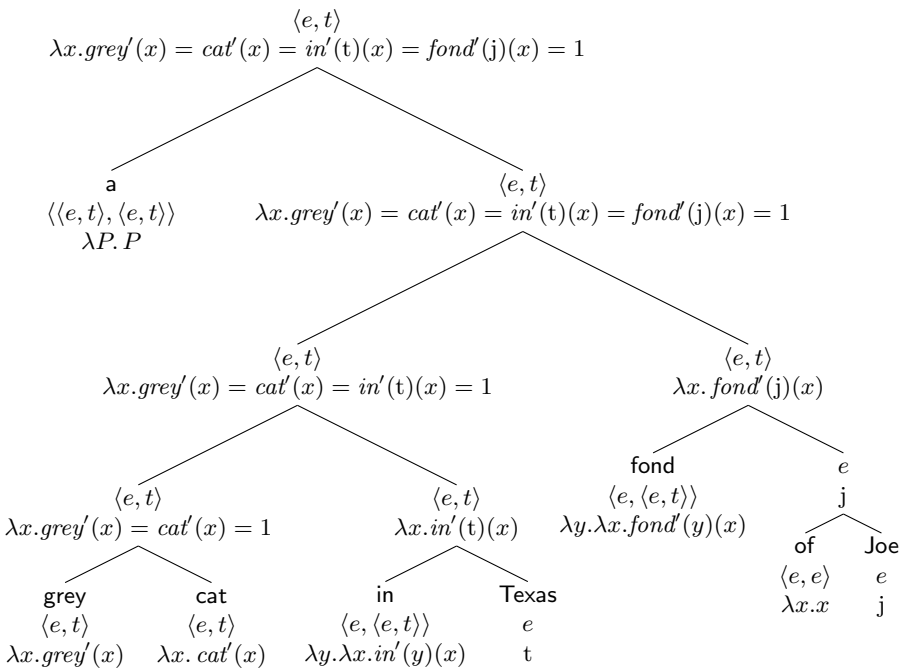


Figure 1: a grey cat in Texas fond of Joe

Notice that the order in which the modifiers combine **makes no difference**. We could have grouped cat in Texas or in Texas fond of Joe as constituents and gotten the same meaning. (Earlier we saw reasons to suppose that order matters.)

Thus, order effects not accounted for:

(4) The visible stars include Capella, Betelgeuse, and Sirius.
   ≠ The stars visible include Capella, Betelgeuse, and Sirius.

We *do* predict ambiguity in other cases:

(5) That's a fake black iPhone.

2.3 Framework

Gives a notion of **type-driven interpretation**

1. Which rule applies is determined by semantic considerations, i.e. we go with whichever option yields a well-defined interpretation.
2. There is never any uncertainty about which rule will apply.
3. Our previous semantics was in fact already type-driven in the sense that which operation applied (forward or backward functional application) was determined by which of the daughter nodes was typed as the function, and which was typed as the argument.

We could keep functional application as our sole method of combination in a few different ways:

1. Positing more complicated meanings for adjectives/PPs, i.e. treating them as functions with modification built in, of type $\langle\langle e,t\rangle,\langle e,t\rangle\rangle$ (here, $grey'$ is the old $\langle e,t\rangle$ meaning):

   $$[\![\mathsf{grey}]\!] = \lambda P_{\langle e,t\rangle}.\lambda x_e.grey'(x) = P(x) = 1$$

   Perhaps there are reasons not to go this way. For one, how to think about sentences like Puffy is grey? Will we need a new meaning for is, too?
2. Positing silent elements in the syntax which do the heavy lifting for us by **coercing** meanings into things with the right type:

   $$[\![\mathsf{mod}_\emptyset]\!] = \lambda P_{\langle e,t\rangle}.\lambda Q_{\langle e,t\rangle}.\lambda x.P(x) = Q(x) = 1$$

   This strategy essentially just moves modification out of the grammar and into the lexicon. Difficult to imagine how we might decide between these two possibilities. (Exercise: try to derive a grey cat in Texas fond of Joe using $\mathsf{mod}_\emptyset$ in lieu of Predicate Modification.)

2.4 Generalized coordination

We saw some potentially problematic instances of conjunction on the last homework (i.e. grammatical, meaningful conjunctions of predicates, transitive verbs, and DPs).

We now have a way to to think about at least some of these case. Perhaps *and* is *ambiguous* between boolean conjunction (type $\langle t,\langle t,t\rangle\rangle$) and predicate conjunction (type $\langle\langle e,t\rangle,\langle\langle e,t\rangle,\langle e,t\rangle\rangle\rangle$):

(6) Uni is orange and in the garage.

(7) Uni licked Puffy and is orange.

(8) Uni meowed and purred.

But how about coordinating transitives?

(9) Uni saw and devoured the burger.

Hm. There seems to be a clear generalization that we do not yet know how to state.

In addition, DP coordination:

(10) John and Bill excel at semantics.

Not obvious how to fold this in. All we know how to do is boolean conjunction and (generalized) predicate conjunction. Perhaps there is a way to view a DP as a function??

Finally, apparent cases of **non-boolean conjunction** exist. The following does not entail that John lifted the piano together, and Bill lifted the piano together:

(11) John and Bill lifted the piano together.

2.5 Issues remain

Our semantics seems to license the following inference (do you see why?):

(12) Tiny is a small elephant.
$\nRightarrow$ Tiny is an elephant, and Tiny is small.

Subsective adjectives? For any f, x:

$$[\![\mathsf{small}]\!] = \lambda f_{\langle e,t\rangle}.\,\lambda x_e.\,x \text{ is small relative to } \{y : f(y) = 1\}$$

Comparison classes offer another option. For any x:

$$[\![\mathsf{small}]\!] = \lambda x_e.\,x \text{ is small relative to the contextual standard } c$$

How far does this get us?

1. Olga is a beautiful dancer (at least on one reading) only if she *dances* beautifully. In other words, the *dimension* of comparison can be regulated by the noun.
2. Superlatives: John read the fewest books. There's no plausible set of things that are the fewest.
3. Privatives: This is a fake gun / John is a former senator.

3 Relative clauses

3.1 Subject relative clauses

Basic data:

(13) Veneeta is a person who took semantics.

(14) Uni is a white cat who purred.

Each of these cases is simple to analyze along the lines of previous modification examples. We get a property in the syntax, and that property combines with the noun by predicate modification (one way or another).

We have a couple options for who: either it denotes an identity function on properties, or it denotes a function that does the work of predicate modification. Nothing much turns on the choice here.

An example derivation is given in Figure 2. This time, we take advantage of some η-equivalences to save space (and assume the semantics of who is vacuous).

3.2 Object relative clauses

Subject relatives are easy to compose.

Object relatives are harder: "gaps" can occur anywhere

(15) The cat [Bill met __] purred.

(16) The cat [Bill showed __ Uni] purred.

(17) The cat [Bill showed Uni __] purred.

More generally, any time a gap isn't at the left edge of the relative clause:

(18) The man [Bill said __ left] was in my class.

So are subjects also a result of movement? Possibly. Parallel issue with wh movement. Subject constituent questions may not require anything special, but object ones (and internal ones) certainly do.

(19) Who (__) saw Uni?

(20) Who did Uni see __?

How to compose up the subject and verb to give a property? What we require:

1. A meaning for the extraction gap the functions as a syntactic and semantic placeholder.
2. A way to use that placeholder to derive a property.

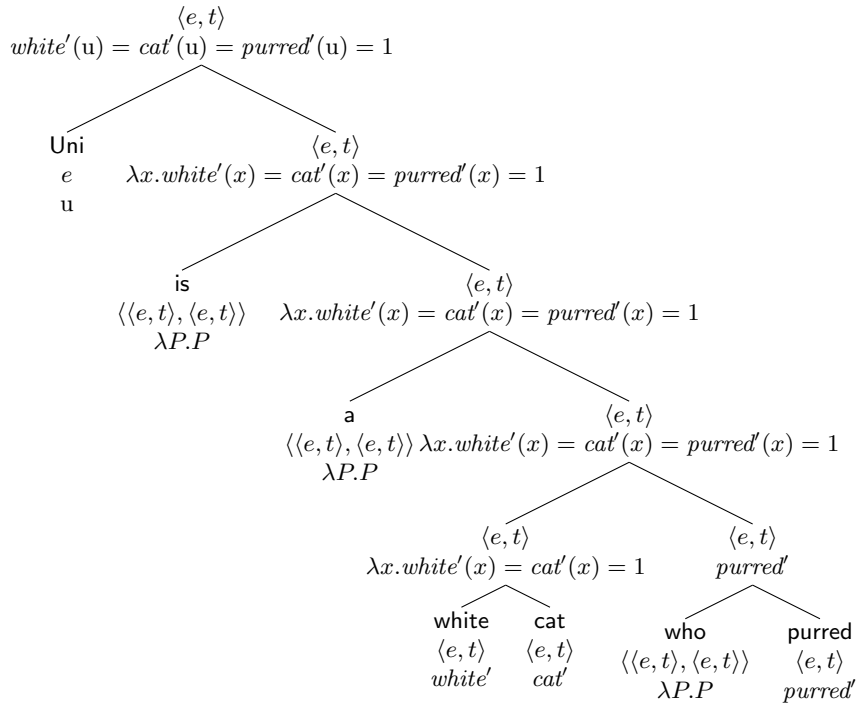


Figure 2: Uni is a white cat who purred.

4 Definite descriptions, partial functions

4.1 Data

Basic cases:

(21) The escalator to Track 7 was working this morning.
Judgment: False

(22) The escalator in 18 Sem is kind of old.
Judgment: ???

A couple data points suggested by these examples:

1. Definite descriptions seem to denote individuals, i.e. they go all the places individual-denoting DPs like **Uni** do.
2. Definite descriptions come with something like a *felicity condition* on

their use. As a first pass, only ok to use when there is a unique individual in the denotation of the complement noun phrase.

4.2 Semantics

A semantics for the. For any property f:

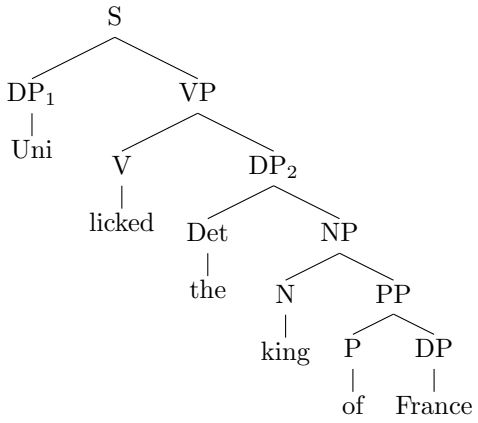
$$[\![\text{the}]\!](f) = \begin{cases} \text{if there is exactly one } x \text{ such that } f(x) = 1, \text{ then that } x \\ \text{undefined otherwise} \end{cases}$$

Alternatively, using our new notation (the following follows H&K's convention for talking about which kinds of arguments a lambda term is defined on):

λf : there is exactly one x such that $f(x) = 1$. the x such that $f(x) = 1$

This semantics is **partial**: it does not assign a meaning to every property f, only those that hold of exactly one individual. For all other cases, it doesn't know what to do; the result is **undefined**.

Undefinedness percolates up the tree. Below, since DP_2 lacks a semantic value, so does any node dominating DP_2, including S.



This is one (popular) way to think about presupposition: sentences whose presuppositions aren't met lack a semantic value. They are neither true nor false. Put differently, a presupposition is a precondition on a sentence being either true or false (as we emphasized in the first week).

Potentially problematic: the following seem to just be False.

(23) The king of France is standing over there

(24) The king of France is a bald Nazi.

Other sorts of undefinedness:

1. Sortal mismatches: Uni is an even divisor of Puffy.
2. Type mismatches: Uni meowed Bill.

5 The syntax of complex DPs

Interpretability constrains our choice of possible analyses of complex DPs. If we go with the first analysis below, it's hard to see how to fold in the relative clause once we've composed up the black cat.

(25) [The black cat] who purred

(26) The [black cat who purred]

Can also see with PP modifiers:

(27) [The book] in the corner

(28) The [book in the corner]

In addition to issues of interpretability, there is an issue of basic interpretive adequacy. Even if we managed to finagle a rule for the first cases, the black cat would denote the single (salient) black cat (and be undefined if there were more than one [salient] black cat).

But we can use the black cat who purred in a context where there's multiple black cats, so long as just one of them purred.

However, we *can* say things like everybody in this room, somebody who met Bill, and so on. This seems to suggest the *other* bracketing. These cases will remain a mystery for now.

6 Next week

Relative clauses, binding, assignment functions

Generalized treatment of DPs:

(29) Every cat grunted.

(30) No cat grunted.

(31) Uni and every pig grunted.

Reading: H&K Ch. 5 (review), and Ch. 6