Continuation semantics i[1]

Patrick D. Elliott[2] *& Martin Hackl*[3]

February 5, 2020

[1] 24.979: Topics in semantics

Getting high: Scope, projection, and evaluation order

[2] pdell@mit.edu

[3] hackl@mit.edu

Add more exercises/questions

1 Roadmap

1.1 Goals for the immediate future

- To be able to understand Barker & Shan's tower notation, and how to translate from towers to flat representations, and vice versa.
- To get a handle on continuation semantics' linear bias, encoded in the composition rules themselves.[4]
- The beginnings of a story for scope islands in terms of obligatory evaluation.

[4] This will be necessary preparation for our discussion of Shan & Barker's (2006) theory of crossover.

1.2 The road we'll travel

- §2: some preliminaries on notational conventions.

Add more goals

2 Some notation conventions

Generally speaking, I'll be assuming Heim & Kratzer 1998 as background, but I'll depart from their notation slightly.

Expressions in the meta-language will be typeset in sans serif.

$$[\![[_{\text{DP}}\ \text{John}]]\!] \coloneqq \underbrace{\mathsf{John}}_{\text{individual}}$$

Seeing as its primitive, we'll treat white-space as function application, e.g.:

(1) $(\lambda x \,.\, \text{left}\ x)\ \text{paul} = \text{left paul}$

Function application associates to the *left*:

(2) $(\lambda x \,.\, \lambda y \,.\, y\ \text{likes}\ x)\ \text{paul sophie} \equiv ((\lambda x \,.\, \lambda y \,.\, y\ \text{likes}\ x)\ \text{paul})\ \text{sophie}$

We'll write *types* in a `fixed width font`. We have our familiar primitive types...

(3) $\texttt{type} \coloneqq \texttt{e} \mid \texttt{t} \mid \texttt{s} \mid \ldots$

...and of course *function types*. Unlike Heim & Kratzer (1998), who use $\langle . \rangle$ as the constructor for a function type, we'll be using the (more standard (outside of linguistics!)) arrow constructor ($\rightarrow$):

(4) $\langle \texttt{e}, \texttt{t} \rangle \equiv \texttt{e} \rightarrow \texttt{t}$

The constructor for function types *associates to the right*:

(5) $\texttt{e} \rightarrow \texttt{e} \rightarrow \texttt{t} \equiv \texttt{e} \rightarrow (\texttt{e} \rightarrow \texttt{t})$

3 *Why bother with continuations?*

Fill in this section

4 *The Partee triangle*

(6) $\text{LIFT}x \coloneqq \lambda k \ .\ k\ x$ $\qquad\qquad \text{LIFT} : \texttt{e} \rightarrow (\texttt{e} \rightarrow \texttt{t}) \rightarrow \texttt{t}$

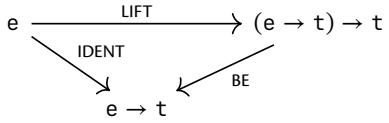
(7) $\text{IDENT}\ x \coloneqq \lambda y \,.\, y = x$ $\qquad\qquad \text{IDENT} : \texttt{e} \rightarrow \texttt{e} \rightarrow \texttt{t}$

(8) $\mathsf{BE}\ Q \coloneqq \lambda x\ .\ Q\ (\lambda y\ .\ y = x)$ $\qquad\qquad$ $\mathsf{BE} : ((\mathtt{e} \to \mathtt{t}) \to \mathtt{t}) \to \mathtt{e} \to \mathtt{t}$

Commutative diagrams

The *Partee triangle* is a *commutative diagram*. We say that a diagram *commutes* if, when there are multiple paths between two points, those paths are equivalent. The equivalence in (10) is therefore expressed by the triangle.

(9) The Partee triangle[5]



[5] Partee 1986

(10) $\mathsf{ident} \equiv \mathsf{BE} \circ (\uparrow)$

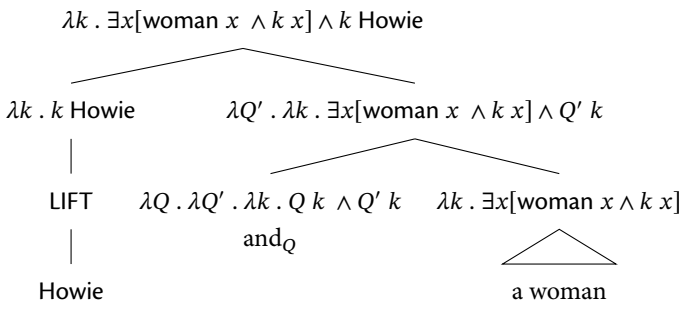
Observation: expressions quantificational and non-quantificational DPs can be coordinated:

(11) [Howie and a woman] entered the club

LIFT allows something that, by virtue of its quantificational nature, is an *inherent* scope-taker, to combine with something that *isn't*:[6]

[6] If you're familiar with Partee & Rooth 2012 you'll notice that the *and* that coordinates quantificational DPs (written here as and_Q), is just the result of applying their *generalized conjunction* rule. We'll return to generalized conjunction, and the connection to coninuations in §8.1.

(12)



4.1 Generalizing the triangle

Based on the way in which LIFT and friends are defined, in theory we could replace e with *any* type. Let's give a more general statement of LIFT and friends as polymorphic functions:

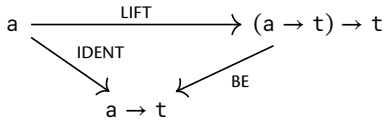
(13) $\mathsf{LIFT} x \coloneqq \lambda k \ . \ k \ x$ $\qquad\qquad \mathsf{LIFT} : \mathtt{a} \to (\mathtt{a} \to \mathtt{t}) \to \mathtt{t}$

(14) $\mathsf{IDENT} \ x \coloneqq \lambda y \ . \ y = x$ $\qquad\qquad \mathsf{IDENT} : \mathtt{a} \to \mathtt{a} \to \mathtt{t}$

(15) $\mathsf{BE} \ Q \coloneqq \lambda x \ . \ Q \ (\lambda y \ . \ y = x)$ $\qquad\qquad \mathsf{BE} : ((\mathtt{a} \to \mathtt{t}) \to \mathtt{t}) \to \mathtt{a} \to \mathtt{t}$

The diagram, of course, still commutes:

(16) The (generalized) Partee triangle



Why might a polymorphic LIFT be useful? Recall that we used a typed instantiation of LIFT in order to allow a quantificational thing to combine with a non-quantificational thing. Polymorphic LIFT allows us to type-lift, e.g., a function that takes multiple arguments:

(17) $\mathsf{LIFT} \ (\lambda x y \ . \ y \ \mathsf{hug} \ x) = \overbrace{\lambda k \ . \ k \ (\lambda x y \ . \ y \ \mathsf{hug} \ x)}^{((\mathtt{e} \to \mathtt{e} \to \mathtt{t}) \to \mathtt{t}) \to \mathtt{t}}$

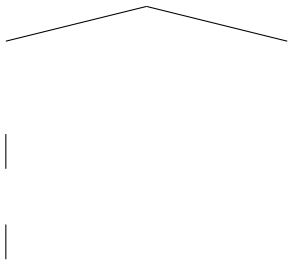
One tantalizing possibility is that this allow us to combine a non-quantificational transitive verb with a quantificational DP.

$((\mathtt{e} \to \mathtt{e} \to \mathtt{t}) \to \mathtt{t}) -$
$\lambda k \,.\, k\ (\lambda x y \,.\, y\ \mathsf{hug}\ x$

LIFT

$\lambda x y \,.\, y\ \mathsf{hug}\ x$

(18)



Unfortunately, assuming the usual inventory of composition rules (i.e., *function application*, *predicate modification*, and *predicate abstraction*), we're stuck.[7] So,

[7] Other, more exotic composition rules such as *restrict* won't help either. Take my word for this!

let's invent a new one.

Here's the intuition we're going to pursue. Let's look again at the types. One way of thinking about what LIFT as follows: it takes an a-type thing and adds a "wrapper". Quantificational DPs, on the other hand come "pre-wrapped".

- LIFT ⟦hug⟧ : ($\boxed{(e \to e \to t)}$ → t) → t
- ⟦a woman⟧ : ($\boxed{e}$ → t) → t

If we look at the wrapped-up types, we see a *function* from individuals, and an individual – namely, two things that can combine via function application.

What we need to accomplish is the following:

- Unwrap lifted *hug*.
- Unwrap the *a woman*.
- Use function application to combine the unwrapped values.
- Finally, wrap the result back up! Think of the quantificational meaning as being like a taco – it isn't really a taco without the wrapper, therefore we don't want to throw the wrapper away.

In order to accomplish this, we'll define a new composition rule: Scopal Function Application (SFA). We're going to define SFA in terms of Function Application (FA); we haven't been explicit about how FA is defined yet, so let's do that now. We'll write FA as the infix operator A.[8]

[8] different ways of making function application bidirectional

(19) Function Application (FA) (def.)

a. f A x ≔ f x — A : (a → b) → a → b

b. x A f ≔ f x — A : a → (a → b) → b

Here, function application is made bidirectional by *overloading* – we've defined forwards and backwards application, and given them the same function name.

Scopal Function Application (SFA)

We'll write SFA as the infix operator S. Note that, since A is overloaded already, and S is defined in terms of A, S gets overloaded too.

(20) Scopal Function Application (SFA) (def.)

$m\ \mathsf{S}\ n \coloneqq \lambda k\ .\ m\ (\lambda a\ .\ n\ (\lambda b\ .\ k\ (a\ \mathsf{A}\ b)))$

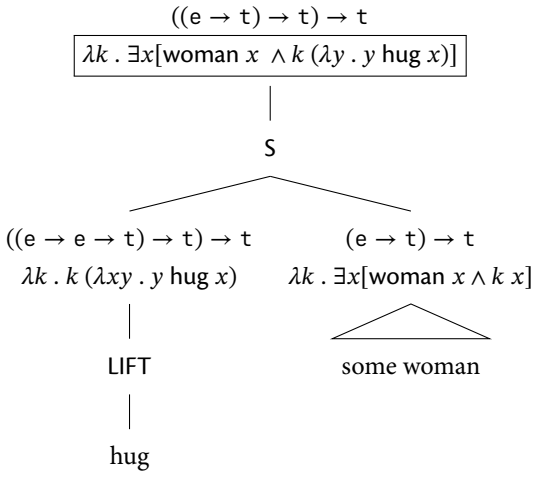
$\mathsf{S} : (((\mathtt{a} \to \mathtt{b}) \to \mathtt{t}) \to \mathtt{t}) \to ((\mathtt{a} \to \mathtt{t}) \to \mathtt{t}) \to (\mathtt{b} \to \mathtt{t}) \to \mathtt{t}$

$\mathsf{S} : ((\mathtt{a} \to \mathtt{t}) \to \mathtt{t}) \to (((\mathtt{a} \to \mathtt{b}) \to \mathtt{t}) \to \mathtt{t}) \to (\mathtt{b} \to \mathtt{t}) \to \mathtt{t}$

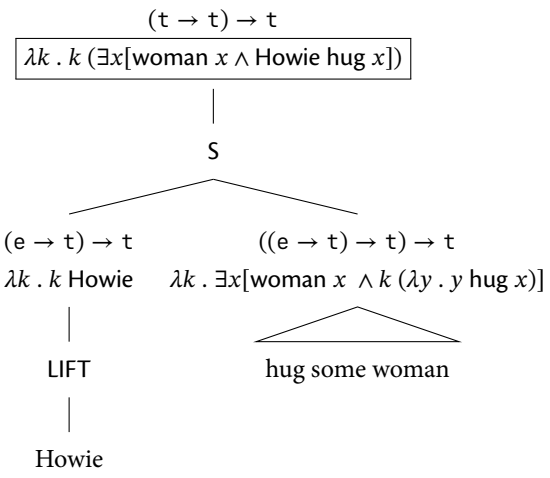
Now, let's illustrate how SFA plus generalized LIFT allows to compose a quantificational thing with non-quantificational things, using a simple example sentence:

(21) Howie hugged some woman.

(22) Step 1: compose *some woman* with LIFT-ed *hug*.



(23) Step 2: compose the resulting VP-denotation with LIFT-ed *Howie*



We're now tantalizingly close to deriving the right kind of object for the sentential meaning (namely, something of type t). Only, what we have is something of type (t → t) → t, i.e., a truth value in a "wrapper". How do we get back the wrapped up value? We saturate the k argument with the identity function. We'll call this operation LOWER.[9]

[9] Here we've given LOWER the maximally polymorphic type compatible with the function definition; in fact, all we need is (LOWER : ((t → t) → t) → t).

(24) LOWER (def.)
LOWER m := m id LOWER : ((a → a) → a) → a

Applying LOWER to the final value in (23) gives us a type t sentential meaning.

(25) LOWER (λk . k ($\exists x$[Howie hug x]))
= (λk . k($\exists x$[Howie hug x])) id
= id ($\exists x$[Howie hug x])
= $\exists x$[Howie hug x]

Let's zoom back out and see how all the pieces fit together by looking at the *graph of the derivation*.

t

LOWER

(t → t) → t

S

(e → t) → t ((e → t) → t) → t

S

LIFT

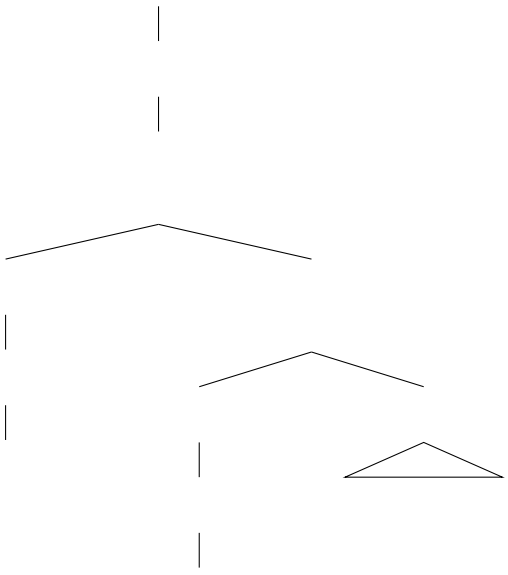
((e → e → t) → t) → t (e → t) → t

Howie

LIFT some woman

hug

(26)



So far, we've provided an account of how quantificational things compose with non-quantificational things, by making use of...

- ...an independently motivated type-shifting rule (LIFT)...
- ...a way to apply LIFT-ed values (S)...
- ...and a way to get an ordinary value back from a LIFT-ed value (LOWER).

This seems pretty nice, but as I'm sure you've noticed, things are quickly going to get pretty cumbersome with more complicated sentences, especially with multiple quantifiers. Before we go any further, let's introduce some notational conveniences.

5 *Towers*

We've been using the metaphor of a *wrapper* for thinking about what LIFT does to an ordinary semantic value. Let's make this a bit more transparent by introducing a new *type constructor* for LIFT-ed values.[10]

(27) $\mathtt{K_t\ a} \coloneqq (\mathtt{a} \to \mathtt{t}) \to \mathtt{t}$

- Quantificational DPs are therefore of type $\mathtt{K_t\ e}$ (inherently).
- LIFT takes something of type $\mathtt{a}$, and lifts it into something of type $\mathtt{K_t\ a}$.

Rather than dealing with *flat* expressions of the simply-typed lambda calculus, which will become increasingly difficult to reason about, we'll follow Barker & Shan 2014 in using *tower notation*.[11]

Let's look again at the meaning of a quantificational DP. The k argument which acts as the *wrapper* is called the *continuation argument*.

(28) $[\![\text{some woman}]\!] \coloneqq \lambda k\ .\ \exists x[\text{woman}\ x\ \wedge k\ x]$

(29) $[\![\text{some woman}]\!] \coloneqq \dfrac{\exists x[\text{woman}\ x \wedge [\,]]}{x}$

[10] A *type constructor* is just a function from a type to a new type – here, it's a rule for taking any type a and returning the type of the corresponding LIFT-ed value.

[11] To my mind, one of Barker & Shan's central achievements is simply the introduction of an accessible notational convention for reasoning about the kinds of lifted meanings we're using here.

(30) $\mathsf{LIFT}\ (\llbracket \text{hug} \rrbracket) = \dfrac{[]}{\lambda xy\ .\ y\ \text{hug}\ x}$

In general:

(31) $\dfrac{f\ []}{x} \coloneqq \lambda k\ .\ f\ (k\ x)$

We can use tower notation for types too:

(32) $\dfrac{\mathsf{b}}{\mathsf{a}} \coloneqq (\mathsf{a} \to \mathsf{b}) \to \mathsf{b}$

We can now redefine our type constructor $\mathsf{K_t}$, and our type-shifting operations using our new, much more concise, tower notation. These will be our canonical definitions from now on. We'll also start abbreviating a LIFT-ed value a as $a^{\uparrow}$ and a LOWER-ed value b as $b^{\downarrow}$.

(33) The continuation type constructor $\mathsf{K_t}$ (def.)

$$\mathsf{K_t\ a} \coloneqq \frac{\mathsf{t}}{\mathsf{a}}$$

(34) LIFT (def.)[12]

$$a^{\uparrow} \coloneqq \frac{[]}{a} \qquad (\uparrow) : \mathsf{a} \to \mathsf{K_t\ a}$$

(35) Scopal Function Application (SFA) (def.)[13]

$$\frac{f\ []}{x}\ \mathsf{S}\ \frac{g\ []}{y} \coloneqq \frac{f\ (g\ [])}{x\ \mathsf{A}\ y} \qquad \mathsf{S} : \mathsf{K_t\ (a \to b) \to K_t\ a \to K_t\ b}$$

(36) LOWER (def.)[14]

$$\left(\frac{f\ []}{p}\right)^{\downarrow} = f\ p \qquad (\downarrow) : \mathsf{K_t\ t \to t}$$

show how lower works in terms of flat lambdas

In order to see the tower notation in action, let's go through an example involving multiple quantifiers, and show how continuation semantics derives the

[12] Thinking in terms of towers, LIFT takes a value a and returns a "trivial" tower, i.e., a tower with an empty top-story.

[13] SFA takes two scopal values – one with a function on the bottom floor, and the other with an argument on the bottom floor – and combines them by (i) doing function application on the bottom floor, and (ii) *sequencing* the scope-takers.

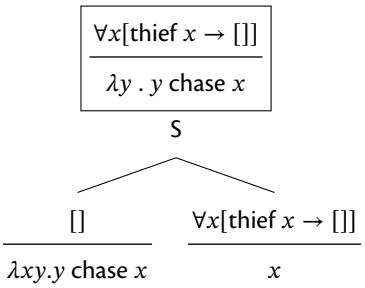
[14] LOWER *collapses the tower*, applying whatever is on the top story to whatever is on the bottom story.

surface scope reading:

(37) Some knight chased every thief.

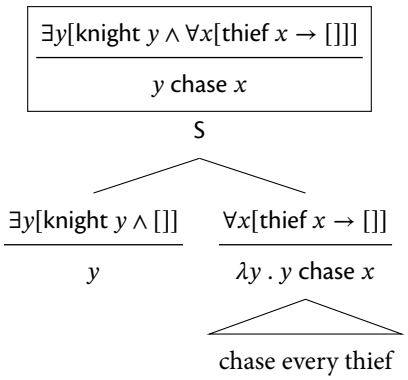
First, we combine *every thief* with lifted *chase* via SFA:

(38)



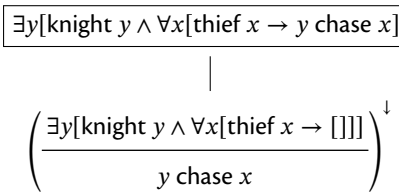
Next, the (boxed) VP value combines with *some knight* via SFA:

(39)

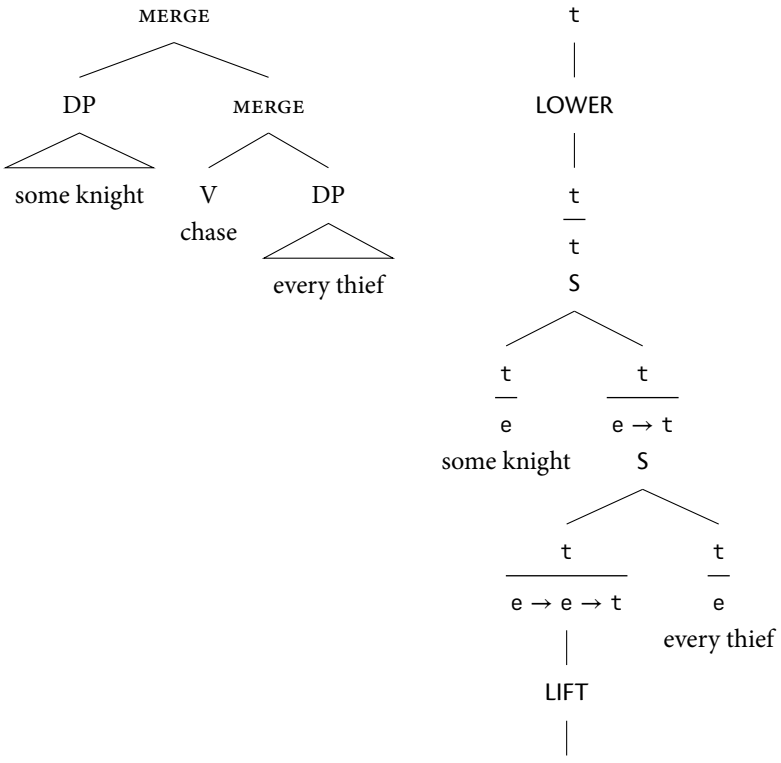


Finally, the resulting tower is collapsed via *lower*:

(40)



Let's zoom out and look at the *graph of the syntactic derivation* alongside the *graph of the semantic derivation*.



6 *Scopal ambiguities*

6.1 Interaction between scope-takers and other operators

Right now, we have a theory which is very good at deriving the surface scope reading of a sentence with multiple quantifiers. It can also derive some ambiguities that arise due to interactions of quantifiers and scopally *immobile* expressions, such as intensional predicates. Consider, e.g., the interaction between a universal quantifier and the desire verb *want*.

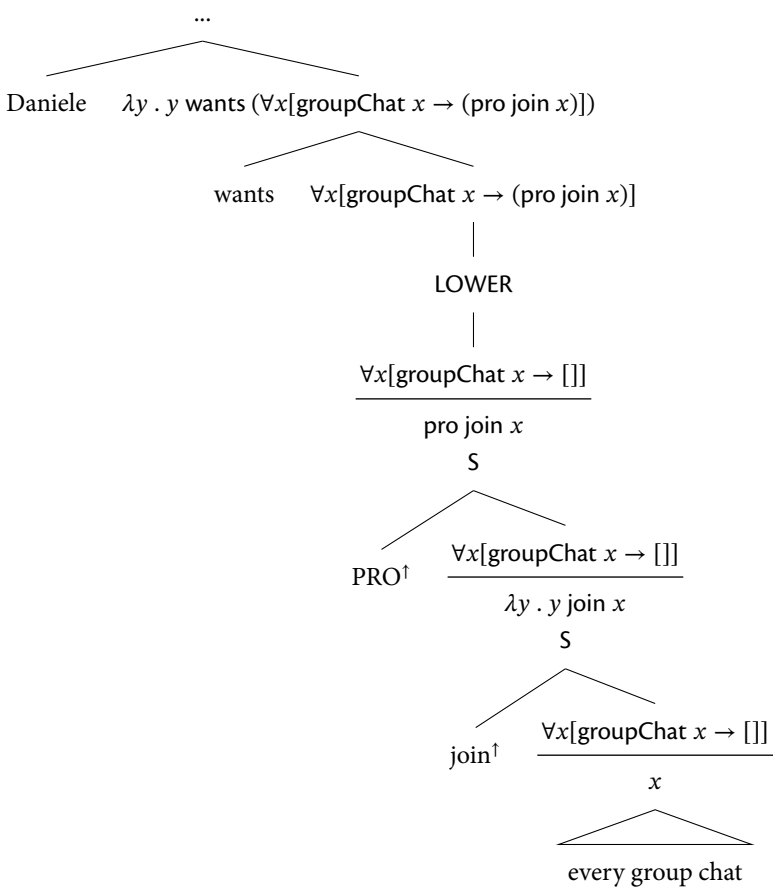
(41) Daniele wants to join every group chat. want > ∀; ∀ > want

(41) is ambiguous: (i) if *every* takes scope below *want*, it is true if Dani has a desire about joining every group chat, (ii) if *every* takes scope over *want*, it is true if every group chat is s.t. Dani has a desire to join in.

We actually already have everything we need in order to account for this. In a nutshell, the two readings correspond to an application of LOWER either below or above the intensional predicate.[15]

[15] For a fully-fledged treatment of the examples below, we would of course need to systematically replace `t` in our fragment to some intensional type.

(42) Daniele wants to join every group chat. want > ∀



(43) Daniele wants to join every group chat. ∀ > want

We can schematize what's going on here by boxing the point of the derivation at which LOWER applies:

- Daniele (wants $\boxed{\text{pro}^{\uparrow}\ \text{S}\ (\text{join}^{\uparrow}\ \text{S}\ \text{everyGroupChat})}^{\downarrow}$)
- $\boxed{\text{Daniele}^{\uparrow}\ \text{S}\ (\text{wants}^{\uparrow}\ \text{S}\ (\text{pro}^{\uparrow}\ \text{S}\ (\text{join}^{\uparrow}\ \text{S}\ \text{everyGroupChat})))}^{\downarrow}$

If you're more familiar with a treatment of scope-taking in terms of quantifier raising, then you can think of LOWER as being the correlate of the landing site of QR; semantic composition proceeds via S up until we encounter the landing site, at which point we switch back to "vanilla" semantic composition via A.

6.2 Scope rigidity

As we've seen however, when we're dealing with multiple scopally *mobile* expressions (such as quantifiers), continuation semantics derives surface scope

readings by default. It is, therefore, well-suited to languages such as German and Japanese, which have been argued to display *scope-rigidity* (modulo semantic reconstruction amongst other potential exceptions).

The following example is from Kuroda (1970). add ref

(44) a. *Dareka-ga subete-no hon-o yonda.*
someone-NOM all-GEN book-ACC read.
"Someone read all the books." ∃ > ∀; ✗ ∀ > ∃

b. *Subete-no hon-o dareka-ga yonda.*
all-GEN book-ACC someone-NOM read.
"Someone read all the books" ∀ > ∃; ∃ > ∀

There are also, of course, famous environments in English where we observe apparent scope-rigidity, such as the double-object construction (scope rigidity in the double-object construction is usually described as *scope freezing*).

(45) a. Daniele sent a syntactician every sticker. ∃ > ∀; ✗ ∀ > ∃
b. Daniele sent a sticker to every syntactician. ∃ > ∀; ∀ > ∃

Bobaljik & Wurmbrand (2012) posit a (violable) economy condition in order to express the preference for surface scope observed in many languages. Bobaljik & Wurmbrand make architectural assumptions that we aren't necessarily committed to here, but the spirit of Scope Transparency (ScoT) is very much in line with a continuation semantics for quantifiers, where surface scope is the default, and inverse scope will only be achievable via additional application of the type-shifting rules posited.

(46) Scope Transparency (ScoT)
If the order of two elements at LF is $A > B$, then the order at PF is $A > B$.

6.3 Deriving inverse scope

Recall that LIFT is a *polymorphic function* – it lifts a value into a trivial tower:

(47) $a^{\uparrow} := \frac{[]}{a}$

Since LIFT is polymorphic, in principle it can apply to any kind of value – even a tower! Let's flip back to lambda notation to see what happens.

(48) $[\![\text{everyone}]\!] \coloneqq \lambda k \,.\, \forall x[k\ x]$

(49) $[\![\text{everyone}]\!]^{\uparrow} = \lambda l \,.\, l\ (\lambda k \,.\, \forall x[k\ x])$

Going back to tower notation, lifting a tower adds a trivial third story:[16] Following Charlow (2014), when we apply LIFT to a tower, we'll describe the operation as *external lift* (although, it's worth bearing in mind that this is really just our original LIFT function).

[16] In fact, via successive application of LIFT, we can generate an $n-$story tower.

One important thing to note is that, when we externally lift a tower, the quantificational part of the meaning always remains on the same story relative to the bottom floor. Intuitively, this reflects the fact that, ultimately, LIFT alone isn't going to be enough to derive quantifier scope ambiguities.

(50) $\left(\dfrac{\forall x[]}{x}\right) = \dfrac{[]}{\dfrac{\forall x[]}{x}}$

Question

Which (if any) of the following bracketings make sense for a three-story tower:

(51) $\dfrac{\left(\dfrac{f\ []}{g\ []}\right)}{x}$

(52) $\dfrac{f\ []}{\left(\dfrac{g\ []}{x}\right)}$

The extra ingredient we're going to need, is the ability to sandwhich an empty story into the *middle* of our tower, pushing the quantificational part of the meaning to the very top. This is *internal lift* (⇑⇑).[17]

(54) *Internal lift* (def.)
   a. $(\Uparrow) : \mathsf{K_t}\ \mathsf{a} \to \mathsf{K_t}\ (\mathsf{K_t}\ \mathsf{a})$
   b. $m^{\Uparrow} \coloneqq \lambda k \,.\, m\ (\lambda x \,.\, k\ x^{\uparrow})$

It's much easier to see what internal lift is doing by using the tower notation. We can also handily compare its effects to those of *external* lift.

[17] I can tell what you're thinking: "seriously? Another *darn* type-shifter? How many of these bad boys are we going to need?!". Don't worry, I got you. Even thought we've defined internal lift here as a primitive operation, it actually just follows from our existing machinery. Concretely, *internal lift* is really just *lifted* LIFT (so many lifts!). Lifted LIFT applies to its argument via S.

(53) $(\mathsf{LIFT}^{\uparrow})\ \mathsf{S}\ \dfrac{f\ []}{x} = \dfrac{\dfrac{f\ []}{[]}}{x}$

(55) *Internal lift* (tower ver.)

$$\left(\frac{f\ []}{x}\right)^{\Uparrow} := \frac{\frac{f\ []}{[]}}{x}$$

(56) *External lift* (tower ver.)

$$\left(\frac{f\ []}{x}\right)^{\uparrow} := \frac{[]}{\frac{f\ []}{x}}$$

Armed with *internal* and *external* lifting operations, we now have everything we need to derive inverse scope. We'll start with a simple example (57).

The trick is: we *internally* lift the quantifier that is destined to take wide scope. That's it!

(57) A boy danced with every girl. $\forall > \exists$

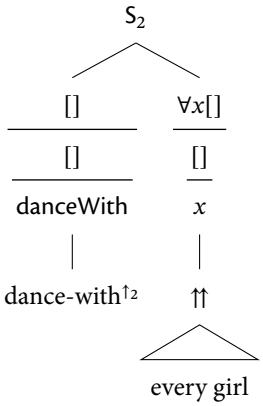
Before we proceed, we need to generalize LIFT and SFA to three-story towers.[18]

(58) $x^{\uparrow_2} := \frac{\frac{[]}{[]}}{x}$

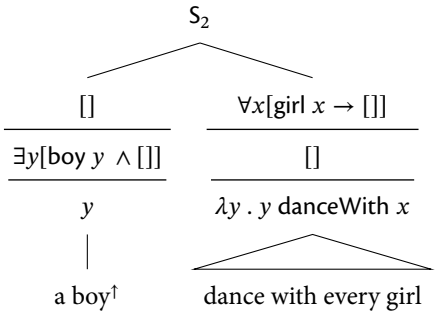
(59) $\frac{f\ []}{m}\ \mathsf{S}_2\ \frac{g\ []}{n} := \frac{f\ (g\ [])}{m\ \mathsf{S}\ n}$

[18] Before you get worries about expanding our set of primitive operations, notice that *3-story lift* is just ordinary lift applied twice. *3-story* SFA is just SFA, but where the bottom story combines via S not A. In fact, we can generalize these operations to $n-$story towers.

(60) Step 1: internally lift *every girl*



(61) Step 2: *ex*ternally lift *a boy*



What we're left with now is a 3-story tower with the universal on the top story and the existential on the middle story. We can collapse the tower by first collapsing the bottom two stories, and then collapsing the result. In order to do this, we'll first define *internal lower*.[19]

[19] Let's again address the issue of expanding our set of primitive operations (in what is becoming something of a theme). Internal lower is just lifted lower, applying via S. In other words:

(62) $m^{\Downarrow} \equiv (\downarrow)^{\uparrow}\ \mathsf{S}\ m$

(63) *Internal lower* (def.) $\left(\dfrac{f\ []}{\dfrac{g\ []}{p}}\right)^{\Downarrow} := \dfrac{f\ []}{\left(\dfrac{g\ []}{x}\right)^{\downarrow}}$

Now we can collapse the tower by doing *internal lower*, followed by *lower*:

(64)

$$\boxed{\forall x[\text{girl } x \rightarrow (\exists y[\text{boy } y \wedge y \text{ danceWith } x])]}$$

$$\downarrow$$

$$\frac{\forall x[\text{girl } x \rightarrow []]}{\exists x[\text{boy } x \wedge y \text{ danceWith } x]}$$

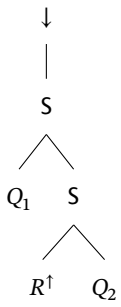
$$\Downarrow$$

$$\frac{\dfrac{\forall x[\text{girl } x \rightarrow []]}{\exists y[\text{boy } y \wedge []]}}{y \text{ danceWith } x}$$

a boy danced with every girl

Great! We've shown how to achieve quantifier scope ambiguities using our new framework. Let's look at the derivations again side-by-side.

(65) Surface scope (schematic derivation)



(66) Inverse scope (schematic derivation)

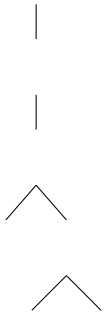
$\downarrow$

$\Downarrow$

S_2

$Q_1^{\uparrow}$ S_2

$R^{\uparrow_2}$ $Q_2^{\uparrow\uparrow}$



There's a couple of interesting things to note here:

- The inverse scope derivation involves more applications of our type-shifting operations – this becomes especially clear if we decompose the complex operations S_2, $\uparrow_2$, $\Uparrow$, and $\Downarrow$.
- In order to derive an inverse scope reading, what was *crucial* was the availability of *internal lift*; the remaining operations, S_2, $\uparrow_2$, $\Downarrow$ only functioned to massage composition for three-story towers.

On the latter point, it's tempting to conjecture that in, e.g., German, Japanese and other languages which "wear their LF on their sleeve", the semantic correlate of *scrambling* is *internal lift*, whereas in scope-flexible languages such as English, internal lift is a freely available operation.[20]

Something something about processing, inverse scope, and derivational complexity

It's worth mentioning, incidentally, that although we collapsed the resulting three-story tower via internal lower followed by lower, we can also define an operation that collapses a three-story tower two an ordinary tower in a different way. Let's call it *join*:[21]

(67) *join* (def.)
$m^{\mu} \coloneqq \lambda k \,.\, m\,(\lambda c \,.\, c\; k)$ $\qquad \mu : \mathsf{K_t}\,(\mathsf{K_t}\;\mathsf{a}) \to \mathsf{K_t}\;\mathsf{a}$

In tower terms, join takes a three-story tower and sequences quantifiers from top to bottom:

(68) $$\left(\begin{array}{c} f\;[\,] \\ \hline g\;[\,] \\ \hline x \end{array}\right)^{\mu} = \begin{array}{c} f\;(g\;[\,]) \\ \hline x \end{array}$$

Doing internal lower on a three-story tower followed by lower is equivalent to doing join on a three-story tower followed by lower (as an exercise, convince yourself of this). However, there's may be a good empirical reason for having internal lower as a distinct operation (and since it's just lifted lower, it "comes for free" in a certain sense).

(69) Daniele wants a boy to dance with every girl. $\qquad \forall > \mathsf{want} > \exists$

[20] To make sense of this, we would of course need to say something more concrete about the relationship between syntax and semantics. For an attempt at marrying continuations to a standard, minimalist syntactic component, see my manuscript *Movement as higher-order structure building*.

[21] Join for three-story towers corresponds directly to the *join* function associated with the continuation monad. For more on continuations from a categorical perspective, see the first appendix.

Arguably, (69) can be true if for every girl x, Daniele has the following desire: *a boy dances with y*. This is the reading on which *every boy* scopes over the intensional verb, and *a boy* scopes below it.

If we have *internal lower*, getting this is easy. We *internally lift every girl* and *externally lift a boy*. Before we reach the intensional verb, we fix the scope of *a boy* by doing internal lower. Now we have an ordinary tower, and we can defer fixing the scope of *every girl* via *lower* until after the intensional verb.

If we only have *join* then the scope of *a boy* and *every girl* may vary amongst themselves, but they should either both scope below *want* or both scope above *want*.

7 *Scope islands and obligatory evaluation*

Inspired by research on *delimited control* in computer science[22], Charlow (2014) develops an interesting take on scope islands couched in terms of continuations.

[22] See, e.g., Danvy & Filinski 1992 and Wadler 1994.

He proposes the following definition:

(70) *Scope islands* (def.)
A *scope island* is a constituent that is subject to *obligatory evaluation*.
(Charlow 2014: p. 90)

By *obligatory evaluation*, we mean that every continuation argument *must* be saturated before semantic computation can proceed. In other words, a scope island is a constituent where, if we have something of type $\mathsf{K}_{\mathsf{t}}\ \mathsf{a}$, we cannot proceed.

One way of thinking about this, is that the presence of an unsaturated continuation argument means that there is some computation that is being deferred until later. Scope islands are constituents at which evaluation is *forced*. As noted by Charlow, this idea bears an intriguing similarity to Chomsky's notion of a *phase*.[23]

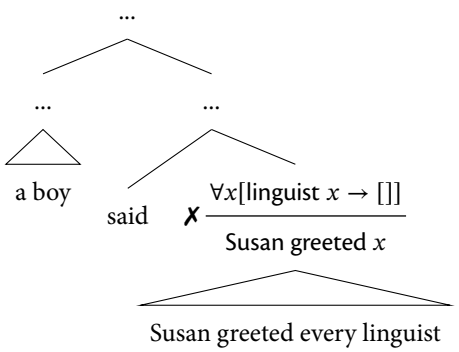
[23] Exploring this parallel in greater depth could make for an interesting term paper topic.

How does this work in practice? A great deal of ink has been spilled arguing that, e.g., a finite clause is a scope island.

(71) A boy said $\overbrace{\boxed{\text{that Susan greeted every linguist}}}^{\text{scope island}}$. $\exists > \forall; \not\forall > \exists$

The derivation of the embedded clause proceeds as usual via lift and SFA.

(72) Scope island with an unevaluated type



(73) Scope island with an evaluated type

...

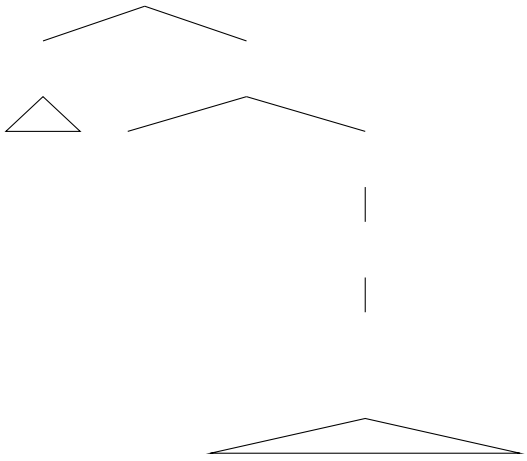
... ...

a boy said ✓ $\forall x$[linguist $x \rightarrow$ Susan greeted x]

↓

$$\frac{\forall x[\text{linguist } x \rightarrow [\,]]}{\text{Susan greeted } x}$$

Susan greeted every linguist



This story leaves a lot of questions unanswered of course:

- Is this just a recapitulation of a representational constraint on quantifier raising?[24]
- Can we give a principled story about islands for overt movement using similar mechanisms? What explains the difference between overt movement and scope taking with respect to locality?[25]

[24] The answer to this question may ultimate be *yes*, in my view.

[25] If we want to give a more general account of phases using this mechanism, we need to give an account of overt movement in terms of continuations, too. See my unpublished ms. *Movement as higher-order structure building* for progress in this direction.

One interesting property of this theory of scope islands is that it goes some way towards explaining why certain expressions, such as indefinites, are able to take *exceptional* scope, given certain assumptions about their semantics.

Charlow (2014) shows that, if we conceive of indefinites as inducing alternatives, scope islands may be obviated by, in a certain sense, pied-piping the scope islands. This strategy won't work for ordinary quantifiers however.[26]

[26] See Demirok (2019) for a parallel story couched in more familiar Heim & Kratzer-esque machinery.

Time permitting, we'll discuss how Charlow accounts for the exceptional scope-taking properties of indefinites using continuations, either next week, or in our discussion of crossover phenomena.

8 *Continuations beyond DPs*

8.1 *Generalized (con/dis)junction*

9 *Indexed continuations*

Let's look again at the "shape" of a continuation type:

(74) $\mathtt{K}_{\mathtt{r}} \coloneqq (\mathtt{a} \to \mathtt{r}) \to \mathtt{r}$

We can consider a more general version of this type (see Wadler 1994).[27]

[27] I'm using the variable names *i* and *r* as mnemonics for *intermediate type* and *return type* respectively.

(75) $\mathtt{K}^{\mathtt{i}}_{\mathtt{r}} \coloneqq (\mathtt{a})$

generalized conjunction as continuised conjunction.

References

Barker, Chris & Chung-chieh Shan. 2014. *Continuations and natural language* (Oxford studies in theoretical linguistics 53). Oxford University Press. 228 pp.

Bobaljik, Jonathan David & Susi Wurmbrand. 2012. Word Order and Scope: Transparent Interfaces and the ¾ Signature. *Linguistic Inquiry* 43(3). 371–421.

Charlow, Simon. 2014. *On the semantics of exceptional scope*.

Charlow, Simon. 2018. A modular theory of pronouns and binding. unpublished manuscript.

Chomsky, Noam. 2001. Derivation by phase. In Kenneth L. Hale & Michael J. Kenstowicz (eds.), *Ken hale: A life in language* (Current Studies in Linguistics 36). Cambridge Massachussetts: The MIT Press.

Danvy, Oliver & Andrzex Filinski. 1992. Representing Control: a Study of the CPS Transformation. *Mathematical Structures in Computer Science* 2(4). 361–391.

Demirok, Ömer. 2019. *Scope theory revisited: Lessons from pied-piping in wh-questions*. Massachussetts Institute of Technology dissertation.

Elliott, Patrick D. 2019. Applicatives for Anaphora and Presupposition. In Kazuhiro Kojima et al. (eds.), *New Frontiers in Artificial Intelligence* (Lecture Notes in Computer Science), 256–269. Cham: Springer International Publishing.

Grove, Julian. 2019. *Scope-taking and presupposition satisfaction*. University of Chicago dissertation.

Heim, Irene & Angelika Kratzer. 1998. *Semantics in generative grammar* (Blackwell textbooks in linguistics 13). Malden, MA: Blackwell. 324 pp.

Kiselyov, Oleg. 2017. Applicative abstract categorial grammars in full swing. In Mihoko Otake et al. (eds.), *New frontiers in artificial intelligence* (Lecture Notes in Computer Science), 66–78. Springer International Publishing.

Mcbride, Conor & Ross Paterson. 2008. Applicative programming with effects. *Journal of Functional Programming* 18(1).

Partee, Barbara. 1986. Noun-phrase interpretation and type-shifting principles. In J. Groenendijk, D. de Jongh & M. Stokhof (eds.), *Studies in discourse representation theory and the theory of generalized quantifiers*, 115–143. Dordrecht: Foris.

Partee, Barbara & Mats Rooth. 2012. Generalized conjunction and type ambiguity. In, Reprint 2012, 361–383. Berlin, Boston: De Gruyter.

Shan, Chung-chieh. 2002. Monads for natural language semantics. *arXiv:cs/0205026*.

Shan, Chung-Chieh & Chris Barker. 2006. Explaining Crossover and Superiority as Left-to-right Evaluation. *Linguistics & Philosophy* 29(1). 91–134.

Wadler, Philip. 1994. Monads and composable continuations. *LISP and Symbolic Computation* 7(1). 39–55.

A Continuations from a categorical perspective

The way we've presented continuation semantics here makes crucial use of three building blocks:

- A type constructor $\mathsf{K_t}$ – a way of getting from any type `a` to an enriched type-space characterizing *continuized* values.
- A function LIFT ($\uparrow$), i.e., a way of lifting a value a into a *trivial* inhabitant of our enriched type-space.
- A composition rule Scopal Function Application (SFA), i.e., an instruction for how to do function application in our enriched type-space.

There's a rich literature in category theory and (derivatively) functional programming on how to characterize this kind of construct, together with law-like properties its components should satisfy in order to qualify as "natural". In fact, as discussed in, e.g., Charlow 2018, Elliott 2019, exactly this kind of general construct is *implicit* in a great deal of semantic theory, including for example theories of pronouns and binding, and theories of focus.

Formally, the triple $(\mathsf{K_t}, \uparrow, \mathsf{S})$ is a special case of an *applicative functor*, a highly influential notion in the literature on functional programming (Mcbride & Paterson 2008); for applications in linguistic semantics see Kiselyov 2017, Charlow 2018, and Elliott 2019.

An applicative functor consists of three components: a *type constructor* `F`, a way of lifting an inhabitant of `a` into an inhabitant of `F a`, called η, and a way of doing *function application* in the enriched type-space `F a`, called $\circledast$.[28] The components of the applicative functor are additionally subject to a number of laws. I give the full definition below:

[28] If you want pronounceable names for these things, η is called *pure* in haskell, and $\circledast$ is called *ap* (short for *application*).

(76) *Applicative functor* (def.)
An *applicative* functor consists of the following three components subject to **homomorphism identity**, **interchange**, and **composition** laws:
a. F : `type` → `type`
b. η : `a` → `F a`
c. $\circledast$: `F (a → b)` → `F a` → `F b`

(77) **Homomorphism**
$f^{\eta} \circledast x^{\eta} \equiv (f\ x)^{\eta}$

(78) **Interchange**
$(\lambda k\ .\ k\ x)^{\eta} \circledast m \equiv m \circledast x^{\eta}$

(79) **Identity**
$id^{\eta} \circledast m \equiv m$

(80) **Composition**
$(\circ)^{\eta} \circledast u \circledast v \circledast w \equiv$
$u \circledast (v \circledast w)$

You can verify for yourselves that the triple $(\mathsf{K_t}, \uparrow, \mathsf{S})$ obeys the applicative laws – we can call it the *continuation applicative*.

A related, more powerful abstraction from the functional programming literature is *monads*. There is a growing body of literature in linguistic semantics that explicitly makes use of monads (see, e.g., Shan 2002, Charlow 2014, Grove 2019, and others). A monad, like an applicative functor, is defined as a triple consisting of a type constructor and two functions. Monads are strictly speaking more powerful than applicative functors; that is to say, if you have a monad you are guaranteed to have an applicative functor, but not vice versa.

(81) *Monad* (def.)
A *monad* consists of the following three components, subject to **associativity** and **identity**.[29]
a. $\mathtt{F} : \mathtt{type} \to \mathtt{type}$
b. $\eta : \mathtt{a} \to \mathtt{F\ a}$
c. $\mu : \mathtt{F\ (F\ a)} \to \mathtt{F\ a}$

(82) **Associativity**
$\mu \circ (\mathsf{map}\ \mu) \equiv \mu \circ \mu$

(83) **Identity**
$\mu\ .\ (\mathsf{map}\ \eta) \equiv \mathsf{join} \circ \eta \equiv id$

We can define join for the continuation monad as follows:

(84) *join* (def.)
a. $\mu : \mathtt{K_t\ (K_t\ a)} \to \mathtt{K_t\ a}$
b. $m^{\mu} \coloneqq \lambda k\ .\ m\ (\lambda c\ .\ c\ k)$

In tower terms, *join* takes a two-level tower and sequences effects from the top story down:

(85) $\dfrac{\dfrac{f\ []}{g\ []}}{x}^{\mu} = \dfrac{f\ (g\ [])}{x}$

Interestingly, it looks like we can define join just in terms of operations from the applicative instance; in other words, the continuations in their monadic guide are no more expressive than continuations in their applicative guise:[30]

[29] I've defined the monad laws here in terms of map, which we haven't discussed. In haskell, this corresponds to `fmap`. In category theory, given a functor $F : C \to D$, this corresponds to the mapping from morphisms in C to morphisms in D supplied by the definition of F. Here, we'll define map as follows: a pair $(\mathtt{F}, \mathsf{map})$ is a *functor*:

P.s. don't worry if you don't know what any of this means! This is only here as a tidbit for interested parties.

[30] Thanks to Julian Grove for discussing this point with me.

(86) $m^{\mu} = m \circ (\uparrow)$

I leave a demonstration of this fact as an exercise.

B LaTeX dojo

Here is the macro I use to typeset towers in LaTeX. Declare this in your preamble. You'll need the `booktabs` and `xparse` packages too.

```
\NewDocumentCommand\semtower{mm}{
  \begin{tabular}[c]{@{\,}c@{\,}}
    \(#1\)
    \\
    \midrule
    \(#2\)
    \\
  \end{tabular}
}
```

A simple two-level tower can now be typeset as follows:

```
$$\semtower{f []}{x}$$
```

Resulting in:

$$\begin{array}{c} f\,[] \\ \hline x \end{array}$$