Continuation semantics i[1]

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

February 8, 2020

[1] 24.979: Topics in semantics

Getting high: Scope, projection, and evaluation order

[2] pdell@mit.edu

[3] hackl@mit.edu

Homework

Please read Barker & Shan 2014, chapters 1 and 4; you can find a pdf on stellar. If you have any questions, feel free to send me them in advance of next week's class, and I'll do my best to address them.
There's also a brief problem set on stellar, due **next Thursday**. This is intended to get you comfortable working with continuations. Like any subsequent p-sets, it won't be graded.

1 Why bother with continuations?

If you're a certain kind of person, the following answer should be sufficient:
playing with new toys is fun.

The longer answer...

- In this seminar, we're going to be investigating mechanisms via which meanings *get high*.
- To put this in a different way, expressions of natural language come with conventionalized meaning components. Often, meaning components end up interpreted in places that don't necessarily correspond to the pronounced position of the expression they're associated with.
- Special cases of this include, but are almost certainly not limited to:
  - Scope-taking (the focus of the first part of this seminar).
  - Presupposition projection (the focus of the latter part of this seminar).
  - Other varieties of "projective" content (e.g., expressives, appositives, etc.).
- We have fairly well-established techniques for dealing with these phenomena, e.g., quantifier raising for scope (May 1977, Heim & Kratzer 1998, etc.), trivalence for presupposition projection (Peters 1979, George 2010, etc.), and conventional implicature for expressives and appositives (Potts 2005, McCready 2010, Gutzmann 2015, etc.) – which aren't without their problems, as we'll see.

- *Continuations* provide a powerful abstraction for modeling meaning components "getting high" in a more general, uniform way.[4]
- This doesn't automatically make them preferable, but if we can get away with reducing our set of theoretical primitives while maintaining the same empirical coverage, we should at least pursue the possibility.
- Continuations were originally developed[5] by computer scientists in order to account for computations that are, in some sense, *delayed* until later on. As such, they're especially well-suited to modelling meaning components, the evaluation of which is *delayed* until later in the derivation.
  - See e.g., Barker & Shan (2014) and related works for a rich literature devoted to applying continuations to *scope taking*.
  - Grove (2019) develops a theory of presupposition projection as a *scopal* phenomenon, using continuations.
  - de Groote (2006) uses continuations to develop a *compositional* dynamic semantics (another important reference is Charlow 2014).
  - In recent work, I've used continuations to model non-local readings of expressive adjectives (Elliott 2019b) and overt movement (Elliott 2019c)
- One of the *design features* of continuation semantics that is of special interest to us is its *built-in left-to-right* bias.
- One thing we'll be thinking about in some depth is how meaning that "gets high", such as scope-takers, interact with anaphoric expressions and other dependees.
- As has been observed for quite some time, interaction between scope and anaphora exhibits a *leftness bias*.

(1) Every student^i's mother adores their$_i$ advisor.

(2) *Their$_i$ advisor adores every student^i's mother.

- The contrast above is an example of a *weak crossover* violation.
- Roughly, weak crossover can be stated as follows:

(3) Weak Crossover (WCO)
Scope may feed binding *unless* the bound expression *precedes* the scope-taker.

- We'd like to explore the possibility, tentatively, that weak crossover isn't just about quantificational scope, but parallel effects can be observed with projective meanings more generally.
- I'll give you a couple of examples here to whet your appetite, although we won't get back to this until quite a bit later in the course.

[4] This goes beyond just *meaning*, and can be generalized to, e.g., phonological and syntactic features. See Elliott 2019c.

[5] Or rather, *discovered*. See, e.g., Danvy & Filinski (1992) and Wadler (1994) for important foundational work on *delimited continuations* – the variety we'll be discussing in this course.

- First, I'd like to suggest that *presupposition projection can feed anaphora*:

(4) Daniel doesn't know Paul has a sister^x, although he has seen her$_x$.

- Focus on the construal where the indefinite takes scope below the intensional verb. Nevertheless, anaphora is possible. The natural way to think about this is as presupposition projection feeding anaphora:[6]

(5) Paul has a sister Daniel doesn't know Paul has a sister, although he has seen her.

- Having established that presupposition projection can feed anaphora, let's see what happens when the pronoun is made to *precede* the presupposition trigger:

(6) #Her$_x$ boss doesn't know that Paul has a sister^x.

- Again, focus on the construal where the indefinite takes scope below the intensional verb. Our example doesn't have the reading indicated. This is surprising given that (a) presuppositions project, (b) presupposition projection can feed anaphora.

- Our hunch is that, what is to blame here is a more general form of WCO – *projective meaning may feed anaphora unless the anaphor precedes the projector.*

- Continuations are sufficiently general, that we can use them to model a wide variety of phenomena. Since they come with a built in linear bias[7], they seem like a prime candidate for modeling linear biases more generally.

[6] A cautionary note: surprisingly, this doesn't follow from standard dynamic theories (such as Heim 1982, Beaver 2001), since presuppositions are themselves static. A natural move is to instead treat presuppositions as themselves dynamic statements; see Elliott & Sudo (2019) for a theory with this character.

[7] And, indeed, arguably one of the most empirically successful accounts of WCO is couched in terms of continuation semantics – see Shan & Barker (2006). We'll discuss this in several weeks time.

2 Plan

This week:

- We're going to develop an understanding of Barker & Shan's tower notation, and how to translate from towers to flat representations, and vice versa.
- We'll get a handle on continuation semantics' linear bias, encoded in the composition rules themselves.[8]
- We'll show how to account for scope ambiguities via a combination of *lowering* and multi-story towers.
- Scope islands in terms of obligatory evaluation.

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

Next week:

- Using continuations to account for generalized con/dis-junction.
- Indexed continuations and a compositional semantics for determiners.
- Continuations and exceptionally scoping indefinites (Charlow 2014).

If we have time:

- Antecedent Contained Deletion.
- Extraposition and scope.
- Expressive adjectives.

3 *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.:

(7) $(\lambda x\ .\ \mathsf{left}\ x)\ \mathsf{paul} = \mathsf{left}\ \mathsf{paul}$

Function application associates to the *left*:

(8) $(\lambda x\ .\ \lambda y\ .\ y\ \mathsf{likes}\ x)\ \mathsf{paul}\ \mathsf{sophie} \equiv ((\lambda x\ .\ \lambda y\ .\ y\ \mathsf{likes}\ x)\ \mathsf{paul})\ \mathsf{sophie}$

> Question
>
> Can the following expression be reduced?
>
> (9) $(\lambda k\ .\ \exists z[k\ z])\ (\lambda x\ .\ \lambda y\ .\ y\ \ \mathsf{likes}\ x)\ \mathsf{Sam}$

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

(10) $\mathtt{type} \coloneqq \mathtt{e} \mid \mathtt{t} \mid \mathtt{s} \mid \ldots$

...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 ($\rightarrow$):

(11) $\langle \mathtt{e}, \mathtt{t} \rangle \equiv \mathtt{e} \rightarrow \mathtt{t}$

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

(12) $\mathtt{e} \rightarrow \mathtt{e} \rightarrow \mathtt{t} \equiv \mathtt{e} \rightarrow (\mathtt{e} \rightarrow \mathtt{t})$

> Question
>
> Which of the following is the correct type for a quantificational determiner?
>
> (13) $[\![\text{every}]\!] : (\mathtt{e} \rightarrow \mathtt{t}) \rightarrow (\mathtt{e} \rightarrow \mathtt{t}) \rightarrow \mathtt{t}$
>
> (14) $[\![\text{every}]\!] : \mathtt{e} \rightarrow \mathtt{t} \rightarrow (\mathtt{e} \rightarrow \mathtt{t}) \rightarrow \mathtt{t}$

4 *The Partee triangle*

The purpose of this section is to show that continuation semantics is, in a certain sense, *already implicit* in the inventory of type-shifters standardly assumed in formal semantics.

(15) $\mathsf{LIFT} x \coloneqq \lambda k \ . \ k \ x$ $\qquad\qquad \mathsf{LIFT} : \mathtt{e} \rightarrow (\mathtt{e} \rightarrow \mathtt{t}) \rightarrow \mathtt{t}$

(16) $\mathsf{IDENT} \ x \coloneqq \lambda y \ . \ y = x$ $\qquad\qquad \mathsf{IDENT} : \mathtt{e} \rightarrow \mathtt{e} \rightarrow \mathtt{t}$

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

Evidence for IDENT:

(18) This man is John.

Evidence for BE:

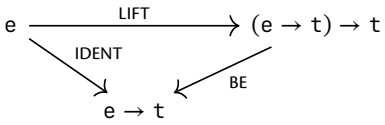
(19) This man is an authority on heavy metal.

We'll come back to LIFT in a moment.

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 (21) is therefore expressed by the triangle.

(20) The Partee triangle[9]



[9] Partee 1986

(21) $\text{ident} \equiv \text{BE} \circ (\uparrow)$

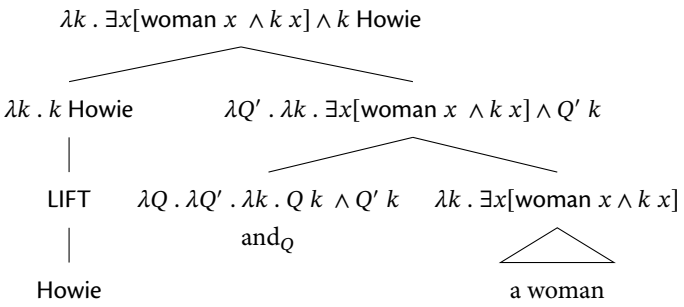
Observation: quantificational and non-quantificational DPs can be coordinated:

(22) [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*:[10]

[10] If you're familiar with Partee & Rooth 1983 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 continuations next week.

(23)



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:

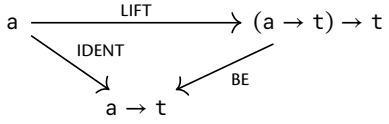
(24) $\textsf{LIFT}\ x \coloneqq \lambda k\ .\ k\ x$ $\qquad\qquad \textsf{LIFT} : \mathtt{a} \to (\mathtt{a} \to \mathtt{t}) \to \mathtt{t}$

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

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

The diagram, of course, still commutes:

(27) The (generalized) Partee triangle



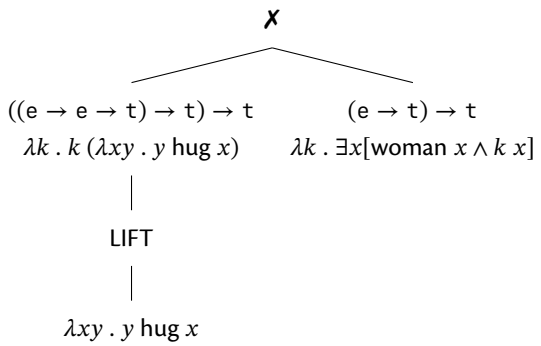
Why might a polymorphic LIFT be useful? Recall that we used a typed instantiation of LIFT in order to allow a quantificational thing to combine with a non-quantificational thing. Polymorphic LIFT allows us to type-lift, e.g., a function that takes multiple arguments:

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

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

(29) Howie hugged a woman

(30)



Unfortunately, assuming the usual inventory of composition rules (i.e., *function application*, *predicate modification*, and *predicate abstraction*), we're stuck.[11] So, **let's invent a new one**.

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

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

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 *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.[12]

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

(31) Function Application (FA) (def.)

a. $f\ \mathsf{A}\ x \coloneqq f\ x$ $\qquad \mathsf{A} : (a \to b) \to a \to b$

b. $x\ \mathsf{A}\ f \coloneqq f\ x$ $\qquad \mathsf{A} : a \to (a \to b) \to b$

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.

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

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

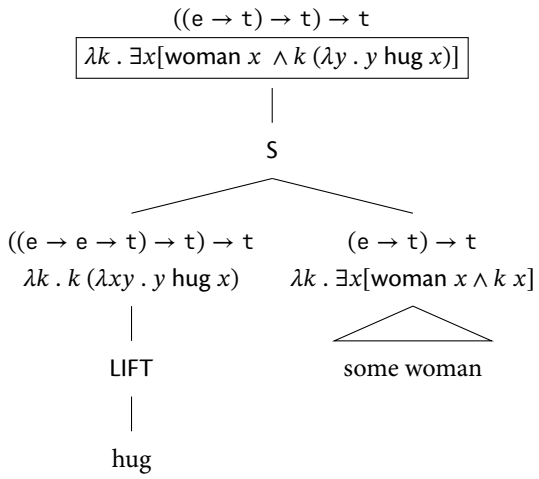
$\mathsf{S} : (((\mathtt{a} \to \mathtt{b}) \to \mathtt{t}) \to \mathtt{t}) \to ((\mathtt{a} \to \mathtt{t}) \to \mathtt{t}) \to (\mathtt{b} \to \mathtt{t}) \to \mathtt{t}$

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

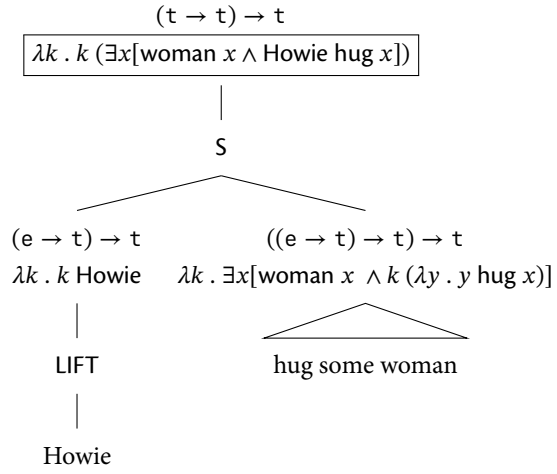
Now, let's illustrate how SFA plus generalized LIFT allows to compose a quantificational thing with non-quantificational things, using a simple example sentence:

(33) Howie hugged some woman.

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



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



At this point, it's worth mentioning that we've **bootstrapped continuation semantics** from an independently motivated type-lifting operation, plus a natural composition rule SFA; continuations aren't scary at all!

The λk argument is standardly referred to as the *continuation argument* in the derivations above. SFA allows the continuation argument to get "passed up".

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 $(\boxed{\mathtt{t}} \to \mathtt{t}) \to \mathtt{t}$, 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.[13]

[13] Here we've given LOWER the maximally polymorphic type compatible with the function definition; in fact, all we need is $(\mathsf{LOWER} : ((\mathtt{t} \to \mathtt{t}) \to \mathtt{t}) \to \mathtt{t})$.

(36) LOWER (def.)
$\mathsf{LOWER}\ m \coloneqq m\ id$ $\qquad\qquad \mathsf{LOWER} : ((\mathtt{a} \to \mathtt{a}) \to \mathtt{a}) \to \mathtt{a}$

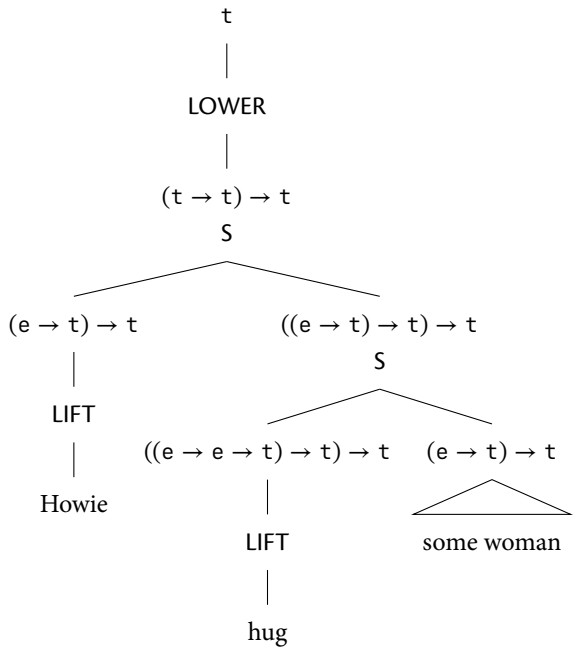
Applying LOWER to the final value in (35) gives us a type t sentential meaning.

(37) $\mathsf{LOWER}\ (\lambda k\ .\ k\ (\exists x[\mathsf{Howie\ hug}\ x]))$

$= (\lambda k\ .\ k(\exists x[\mathsf{Howie\ hug}\ x]))\ id$	by def.
$= id\ (\exists x[\mathsf{Howie\ hug}\ x])$	reduce
$= \exists x[\mathsf{Howie\ hug}\ x]$	reduce

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

(38)



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.[14]

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

[14] 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.

- Quantificational DPs are therefore of type $\mathtt{K_t\ e}$ (inherently).
- LIFT takes something of type 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*.[15,16]

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

[15] 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.

[16] It's important to bear in mind that towers are just *syntactic sugar* for flat lambda expressions; we should always be able to translate back from towers to lambda expressions. Towers have no privileged theoretical status, unlike, e.g., Discourse Representation Structures, but are merely abbreviations.

(40) Ordinary quantifier meanings:
$[\![\text{some woman}]\!] \coloneqq \lambda k\ .\ \exists x[\mathsf{woman}\ x\ \wedge k\ x]$

(41) Quantifiers using tower notation:
$$[\![\text{some woman}]\!] \coloneqq \frac{\exists x[\mathsf{woman}\ x \wedge [\,]]}{x}$$

(42) Lifted meanings using tower notation:
$$\mathsf{LIFT}\ ([\![\text{hug}]\!]) = \frac{[\,]}{\lambda xy\ .\ y\ \mathsf{hug}\ x}$$

In general:

(43) Tower notation (def.)

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

We can use tower notation for types too:

(44) Tower types (def.)

$$\frac{\mathsf{b}}{\mathsf{a}} \coloneqq (\mathsf{a} \to \mathsf{b}) \to \mathsf{b} \qquad \equiv \mathsf{K_b\ a}$$

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 $b^{\downarrow}$.

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

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

(46) LIFT (def.)[17]

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

(47) Scopal Function Application (SFA) (def.)[18]

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

(48) LOWER (def.)[19]

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

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

[18] 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.

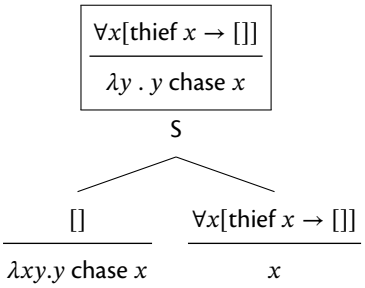
[19] LOWER *collapses the tower*, by saturating the continuation argument with the identity function.

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 surface scope reading:

(49) Some knight chased every thief. $\exists > \forall$

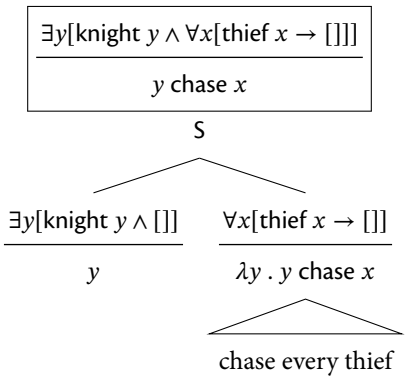
First, we combine *every thief* with lifted *chase* via SFA:

(50)



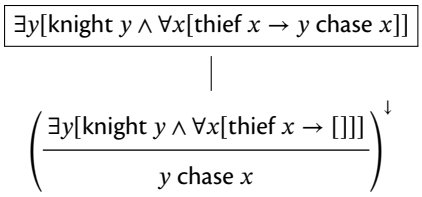
Next, the (boxed) VP value combines with *some knight* via SFA:

(51)

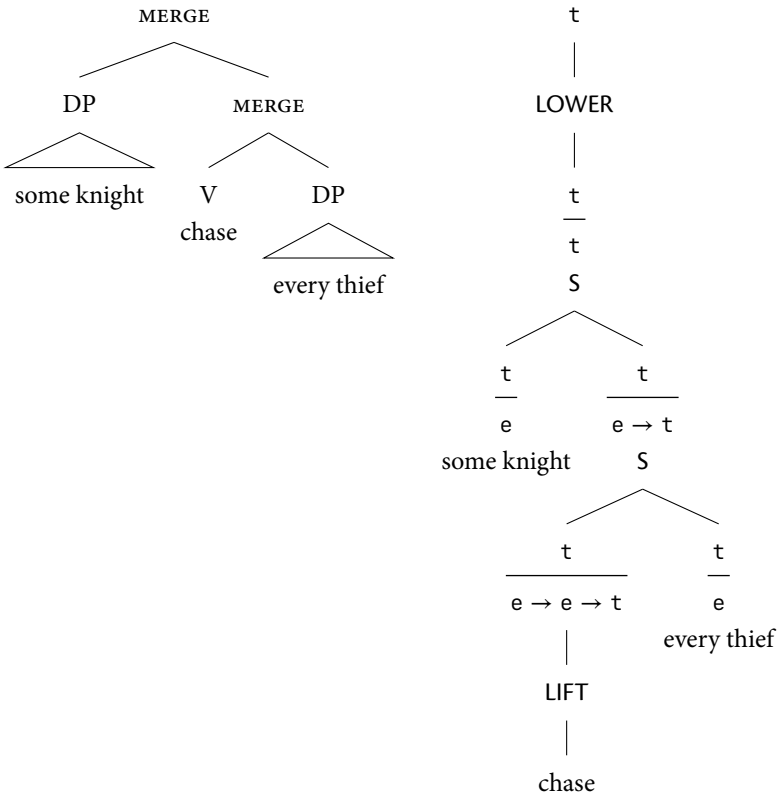


Finally, the resulting tower is collapsed via *lower*:

(52)



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



Type-shifting operations are interleaved with syntactic operations. In order to restrict this system, it seems natural to place an economy constraint on S and LIFT.[20]

[20] This is very tentative. Working out exactly how the system outlined here should be restricted in terms of economy would make a great student project.

(53) Economy condition on type-shifting
MERGE in the syntax is interpreted as A/S in the semantics, whichever is well-typed. LIFT may apply freely to the extent that it is necessary for the derivation to proceed.

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*.

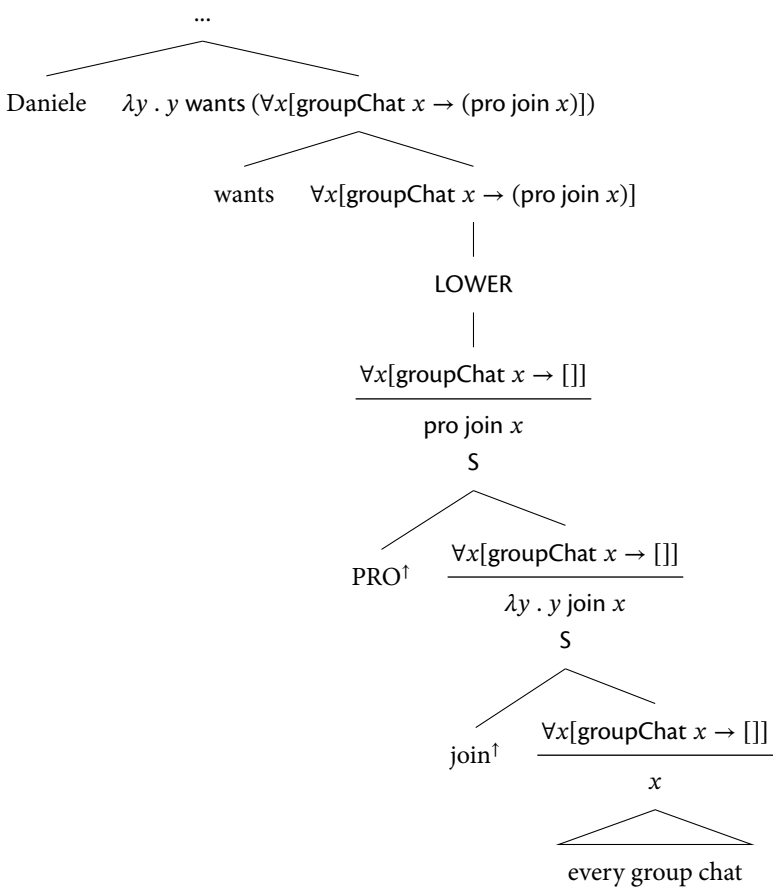
(54) Daniele wants to join every group chat. want > ∀; ∀ > want

(54) 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.

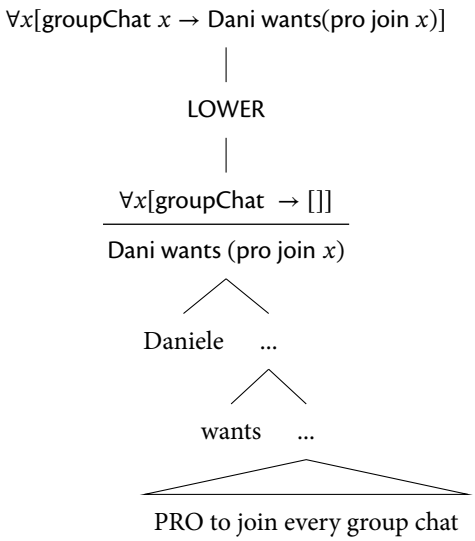
We actually already have everything we need in order to account for this. In a nutshell, the two readings correspond to an application of LOWER either below or above the intensional predicate.[21]

[21] 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.

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



(56) 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).

(57) a. *Dareka-ga subete-no hon-o yonda.*
someone-NOM all-GEN book-ACC read.
"Someone read all the books." $\exists > \forall; \cancel{\forall} > \exists$

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

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*).

(58) a. Daniele sent a syntactician every picture. $\exists > \forall; \cancel{\forall} > \exists$
b. Daniele sent a picture to every syntactician. $\exists > \forall; \forall > \exists$

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.

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

We still need to account for the availability of *inverse scope readings* in English, and other languages however. We'll do this using *multi-story towers*.

7 Next week

Hopefully you're starting to get a feel for what continuations can accomplish with very few primitives. Next week, we'll pick up where we left off, with a discussion of:

- Inverse scope.
- Generalized con/dis-junction.
- *Indexed* continuations and a semantics for determiners.
- Continuations and exceptional scope

After that, Martin will take over with a re-assessment of QR.

References

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

Beaver, David I. 2001. *Presupposition and assertion in dynamic semantics* (Studies in logic, language, and information). Stanford, California: CSLI. 314 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.

de Groote, Philippe. 2006. Proceedings of SALT 16 16. 1–16.

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

Elliott, Patrick D. 2019a. 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.

Elliott, Patrick D. 2019b. *Fuck* compositionality. Slides from an invited talk at the DGfS workshop *Encoding emotive attitudes in non-truth-conditional meaning*.

Elliott, Patrick D. 2019c. Overt movement as higher-order structure building. unpublished manuscript. Leibniz-Zentrum Allgemeine Sprachwissenschaft.

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

Gutzmann, Daniel. 2015. *Use-conditional meaning: studies in multidimensional semantics*. First edition (Oxford Studies in Semantics and Pragmatics 6). OCLC: ocn894201804. Oxford, United Kingdom: Oxford University Press. 304 pp.

Heim, Irene. 1982. *The semantics of definite and indefinite noun phrases*. 2011 edition - typesetting by Anders J. Schoubye and Ephraim Glick. University of Massachusetts - Amherst 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.

May, Robert. 1977. *The grammar of quantification*. Massachussetts Institute of Technology dissertation.

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

McCready, Elin. 2010. Varieties of conventional implicature. *Semantics and Pragmatics* 3(0). 8-1–57.

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. 1983. Generalized conjunction and type ambiguity. In, Reprint 2012, 361–383. Berlin, Boston: De Gruyter.

Potts, Christopher. 2005. *The logic of conventional implicatures* (Oxford Studies in Theoretical Linguistics 7). OCLC: ocm56964776. Oxford: Oxford University Press. 246 pp.
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 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 2019a, 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 2019a.

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 $\circledast$.[22] The components of the applicative functor are additionally subject to a number of laws. I give the full definition below:

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

(60) *Applicative functor* (def.)
An *applicative* functor consists of the following three components subject to **homomorphism identity**, **interchange**, and **composition** laws:
a. $F : \texttt{type} \to \texttt{type}$
b. $\eta : \texttt{a} \to \texttt{F a}$
c. $\circledast : \texttt{F (a} \to \texttt{b)} \to \texttt{F a} \to \texttt{F b}$

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

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

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

(64) **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.

(65) *Monad* (def.)
A *monad* consists of the following three components, subject to **associativity** and **identity**.[23]
a. $\texttt{F} : \texttt{type} \to \texttt{type}$
b. $\eta : \texttt{a} \to \texttt{F a}$
c. $\mu : \texttt{F (F a)} \to \texttt{F a}$

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

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

We can define join for the continuation monad as follows:

(68) *join* (def.)
a. $\mu : \mathsf{K_t}\ (\mathsf{K_t}\ \texttt{a}) \to \mathsf{K_t}\ \texttt{a}$
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:

[23] 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 $(\texttt{F}, \mathsf{map})$ 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.

$$\frac{\frac{f\ []}{g\ []}}{x}^{\mu} = \frac{f\ (g\ [])}{x} \tag{69}$$

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:[24]

$$m^{\mu} = m \circ (\uparrow) \tag{70}$$

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

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:

$$\frac{f\ []}{x}$$