Continuation semantics i[1]

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

February 6, 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 **barkerShan2015**, 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**, **heimKratzer1998**, etc.), trivalence for presupposition projection (Peters 1979, George 2010, etc.), and conventional implicature for expressives and appositives (**potts2005**, **mccready2010a**, **gutzmann2015**, 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]

[4] This goes beyond just *meaning*, and can be generalized to, e.g., phonological and syntactic features. See **elliott2019movement**.

- 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.

[5] Or rather, *discovered*. See, e.g., **danvyFilinski1992** and **wadler1994** for important foundational work on *delimited continuations* – the variety we'll be discussing in this course.

  - See e.g., **barkerShan2015** and related works for a rich literature devoted to applying continuations to *scope taking*.
  - **grove2019** develops a theory of presupposition projection as a *scopal* phenomenon, using continuations.
  - **deGroote2006** uses continuations to develop a *compositional* dynamic semantics (another important reference is **Charlowc**).
  - In recent work, I've used continuations to model non-local readings of expressive adjectives (**elliott-fuck**) and overt movement (**elliott2019movement**)

- 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) **wco! (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.

- 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]

[6] A cautionary note: surprisingly, this doesn't follow from standard dynamic theories (such as **heim1982**, **beaver_presupposition_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.

(5) $\boxed{\text{Paul has a sister}}$ 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.

[7] And, indeed, arguably one of the most empirically successful accounts of WCO is couched in terms of continuation semantics – see **shanBarker2006**. We'll discuss this in several weeks time.

2 *Plan*

This week:

- We're going to develop an understanding of **barkerShan2015**'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]

[8] This will be necessary preparation for our discussion of **shanBarker2006**'s (**shanBarker2006**) theory of crossover.

- 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.

Next week:

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

If we have time:

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

3 *Some notation conventions*

Generally speaking, I'll be assuming **heimKratzer1998** 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) $(\lambda x \mathbin{.} \mathsf{left}\ x)\ \mathsf{paul} = \mathsf{left}\ \mathsf{paul}$

Function application associates to the *left*:

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

Question

Can the following expression be reduced?

(9) $(\lambda k \mathbin{.} \exists z[k\ z])\ (\lambda x \mathbin{.} \lambda y \mathbin{.} y\ \ \mathsf{likes}\ x)\ \mathsf{Sam}$

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

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

...and of course *function types*. Unlike **heimKratzer1998**, 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) $\langle \texttt{e}, \texttt{t} \rangle \equiv \texttt{e} \rightarrow \texttt{t}$

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

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

Question

Which of the following is the correct type for a quantificational determiner?

(13) $[\![\text{every}]\!] : (\texttt{e} \rightarrow \texttt{t}) \rightarrow (\texttt{e} \rightarrow \texttt{t}) \rightarrow \texttt{t}$

(14) $[\![\text{every}]\!] : \texttt{e} \rightarrow \texttt{t} \rightarrow (\texttt{e} \rightarrow \texttt{t}) \rightarrow \texttt{t}$

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

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

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

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 (**??**) is therefore expressed by the triangle.

(20) The Partee triangle[9]

$$\begin{tikzcd}
\mathtt{e} \arrow[rr, "\mathsf{LIFT}"] \arrow[dr, "\mathsf{IDENT}"'] & & (\mathtt{e} \to \mathtt{t}) \to \mathtt{t} \arrow[dl, "\mathsf{BE}"] \\
& \mathtt{e} \to \mathtt{t} &
\end{tikzcd}$$

[9] **partee1986**

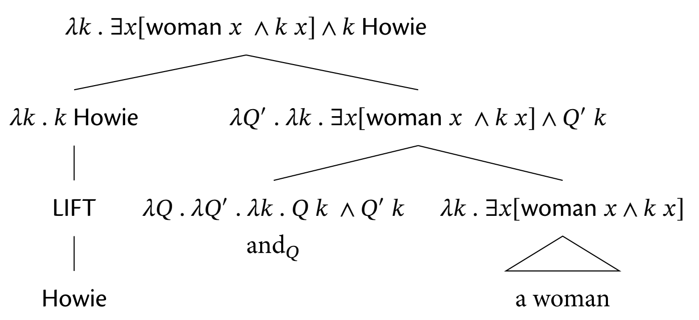
(21) $\mathsf{ident} \equiv \mathsf{BE} \circ (\uparrow)$

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 **parteeRooth** 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)

$\lambda k \,.\, \exists x[\text{woman } x \wedge k\, x] \wedge k \text{ Howie}$
$\lambda k \,.\, k \text{ Howie}$
$\lambda Q' \,.\, \lambda k \,.\, \exists x[\text{woman } x \wedge k\, x] \wedge Q'\, k$
LIFT
$\lambda Q \,.\, \lambda Q' \,.\, \lambda k \,.\, Q\, k \wedge Q'\, k$
and_Q
$\lambda k \,.\, \exists x[\text{woman } x \wedge k\, x]$
Howie
a woman

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

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

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

The diagram, of course, still commutes:

(27) The (generalized) Partee triangle

$$\begin{tikzcd}
\texttt{a} \arrow[rr, "\textsf{LIFT}"] \arrow[dr, "\textsf{IDENT}"'] & & (\texttt{a} \to \texttt{t}) \to \texttt{t} \arrow[dl, "\textsf{BE}"] \\
& \texttt{a} \to \texttt{t} &
\end{tikzcd}$$

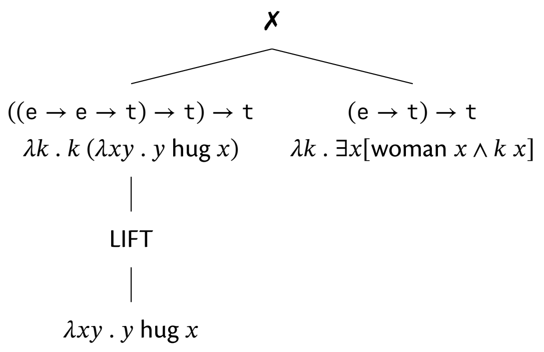
Why might a polymorphic LIFT be useful? Recall that we used a typed instantiation of LIFT in order to allow a quantificational thing to combine with a non-quantificational thing. Polymorphic LIFT allows us to type-lift, e.g., a function that takes multiple arguments:

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

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

(29) Howie $\boxed{\text{hugged a woman}}$

(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: **sfa!** (**SFA!**). We're going to define **SFA!** in terms of **fa!** (**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) **fa!** (**FA!**) (def.)

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

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

sfa! (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.

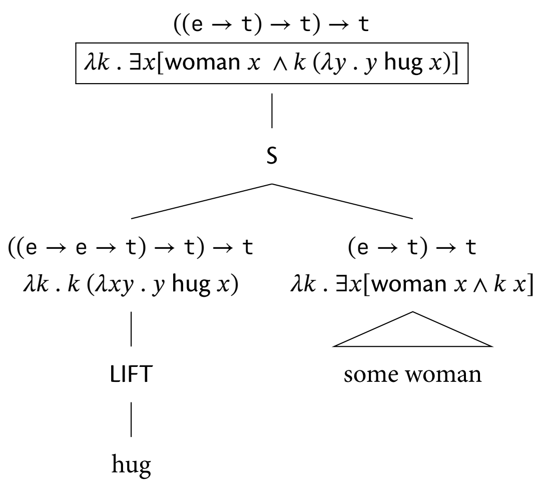
(32) **sfa!** (**SFA!**) (def.)

$$m \mathsf{S} n := \lambda k \,.\, m\, (\lambda a \,.\, n\, (\lambda b \,.\, k\, (a \mathsf{A} b)))$$
$$\mathsf{S} : (((\mathtt{a} \to \mathtt{b}) \to \mathtt{t}) \to \mathtt{t}) \to ((\mathtt{a} \to \mathtt{t}) \to \mathtt{t}) \to (\mathtt{b} \to \mathtt{t}) \to \mathtt{t}$$
$$\mathsf{S} : ((\mathtt{a} \to \mathtt{t}) \to \mathtt{t}) \to (((\mathtt{a} \to \mathtt{b}) \to \mathtt{t}) \to \mathtt{t}) \to (\mathtt{b} \to \mathtt{t}) \to \mathtt{t}$$

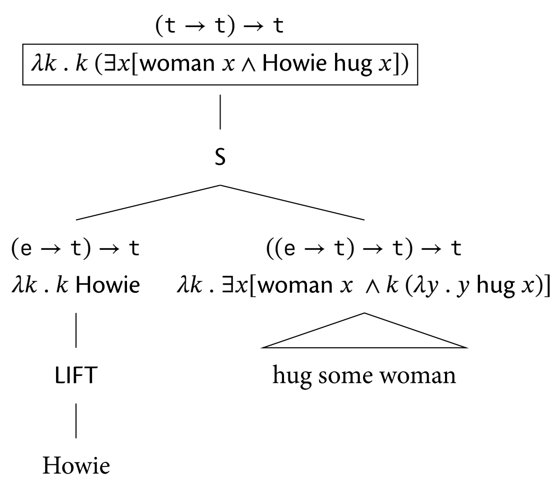
Now, let's illustrate how **SFA!** plus generalized LIFT allows to compose a quantificational thing with non-quantificational things, using a simple example sentence:

(33) Howie hugged some woman.

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


$((\mathtt{e} \to \mathtt{t}) \to \mathtt{t}) \to \mathtt{t}$
$\boxed{\lambda k \,.\, \exists x[\mathsf{woman}\ x \wedge k\, (\lambda y \,.\, y\ \mathsf{hug}\ x)]}$
S
$((\mathtt{e} \to \mathtt{e} \to \mathtt{t}) \to \mathtt{t}) \to \mathtt{t}$
$\lambda k \,.\, k\, (\lambda xy \,.\, y\ \mathsf{hug}\ x)$
LIFT
hug
$(\mathtt{e} \to \mathtt{t}) \to \mathtt{t}$
$\lambda k \,.\, \exists x[\mathsf{woman}\ x \wedge k\ x]$
some woman

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


$(\mathtt{t} \to \mathtt{t}) \to \mathtt{t}$
$\boxed{\lambda k \,.\, k\, (\exists x[\mathsf{woman}\ x \wedge \mathsf{Howie\ hug}\ x])}$
S
$(\mathtt{e} \to \mathtt{t}) \to \mathtt{t}$
$\lambda k \,.\, k\ \mathsf{Howie}$
LIFT
Howie
$((\mathtt{e} \to \mathtt{t}) \to \mathtt{t}) \to \mathtt{t}$
$\lambda k \,.\, \exists x[\mathsf{woman}\ x \wedge k\, (\lambda y \,.\, y\ \mathsf{hug}\ x)]$
hug some woman

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 $\mathtt{t}$). 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 $(\text{LOWER} : ((\mathtt{t} \rightarrow \mathtt{t}) \rightarrow \mathtt{t}) \rightarrow \mathtt{t})$.

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

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

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

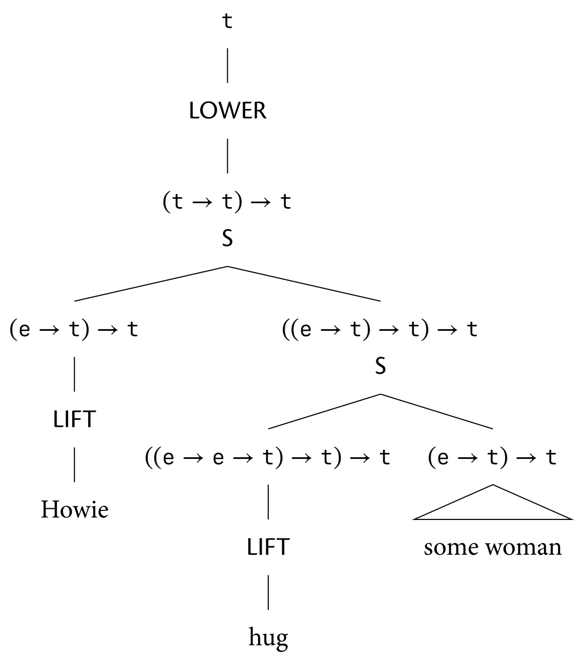
$= (\lambda k\ .\ k(\exists x[\text{Howie hug}\ x]))\ id$ by def.

$= id\ (\exists x[\text{Howie hug}\ x])$ reduce

$= \exists x[\text{Howie hug}\ x]$ 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]

[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.

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

- 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 **barkerShan2015** in using *tower notation*.[15,16]

[15] To my mind, one of **barkerShan2015**'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.

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

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

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

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

In general:

(43) Tower notation (def.)

$$\frac{f\,[]}{x} := \lambda k\,.\,f\,(k\,x)$$

We can use tower notation for types too:

(44) Tower types (def.)

$$\frac{\mathsf{b}}{\mathsf{a}} := (\mathsf{a} \to \mathsf{b}) \to \mathsf{b} \qquad \equiv \mathsf{K}_\mathsf{b}\ \mathsf{a}$$

We can now redefine our type constructor 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 K_t (def.)

$$\mathsf{K}_\mathsf{t}\ \mathsf{a} := \frac{\mathsf{t}}{\mathsf{a}}$$

(46) LIFT (def.)[17]

$$a^{\uparrow} := \frac{[]}{a} \qquad (\uparrow) : \mathsf{a} \to \mathsf{K}_\mathsf{t}\ \mathsf{a}$$

[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.

(47) **sfa!** (**SFA!**) (def.)[18]

$$\frac{f\,[]}{x}\ \mathsf{S}\ \frac{g\,[]}{y} := \frac{f\,(g\,[])}{x\ \mathsf{A}\ y} \qquad \mathsf{S} : \mathsf{K}_\mathsf{t}\ (\mathsf{a} \to \mathsf{b}) \to \mathsf{K}_\mathsf{t}\ \mathsf{a} \to \mathsf{K}_\mathsf{t}\ \mathsf{b}$$

[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.

(48) LOWER (def.)[19]

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

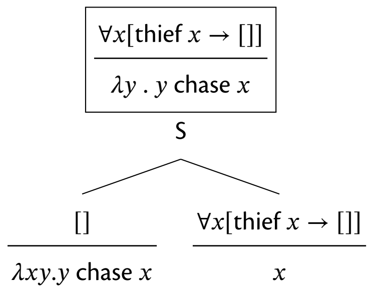
[19] LOWER *collapses the tower*, by saturating the continuation argument with the identity function.

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. $\exists > \forall$

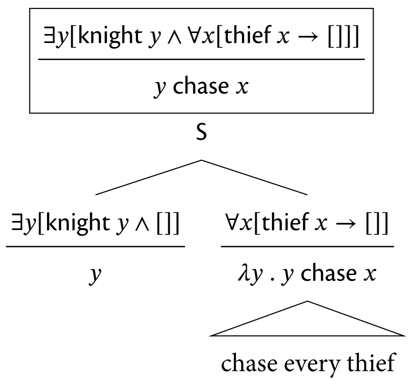
First, we combine *every thief* with lifted *chase* via **SFA!**:

(50)



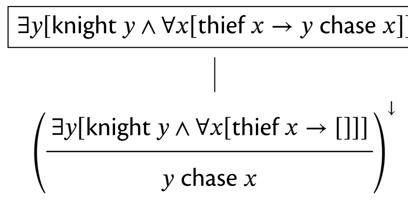
Next, the (boxed) VP value combines with *some knight* via **SFA!**:

(51)

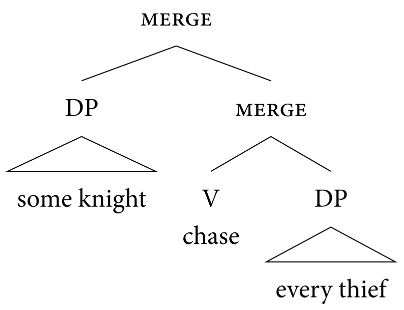


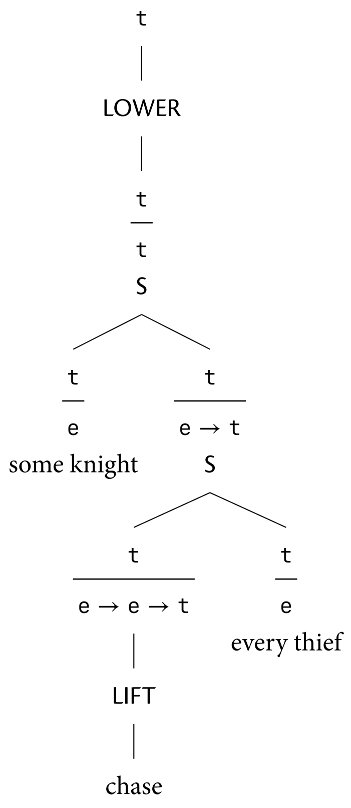
Finally, the resulting tower is collapsed via *lower*:

(52)


∃y[knight y ∧ ∀x[thief x → y chase x]]
∃y[knight y ∧ ∀x[thief x → []]]
y chase x
↓

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*.

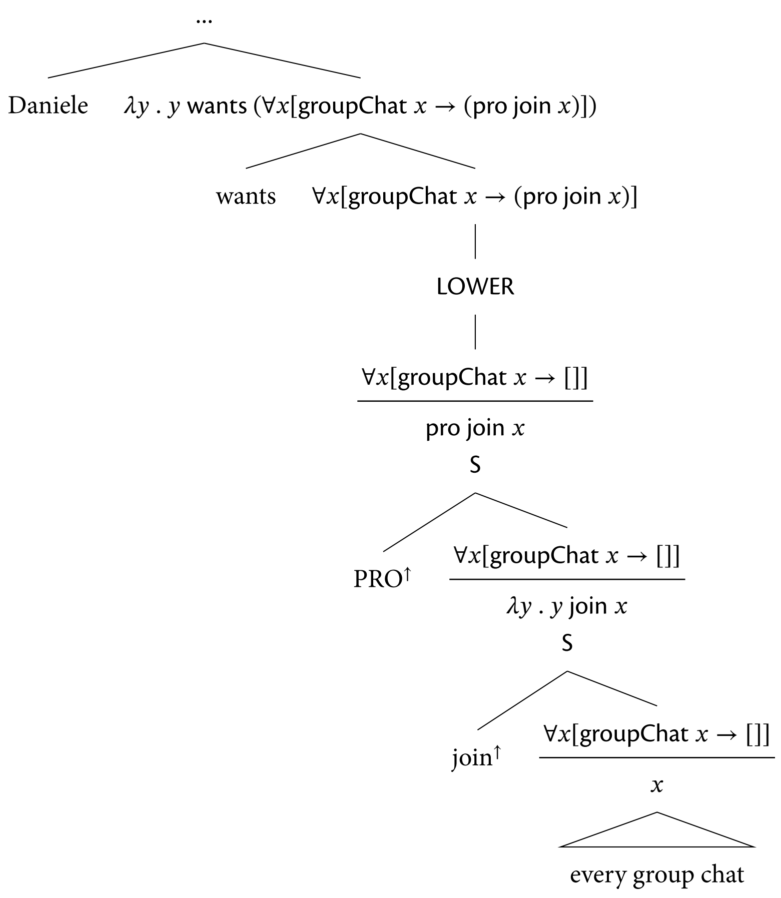
(54) Daniele wants to join every group chat. $\text{want} > \forall; \forall > \text{want}$

(**??**) 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.

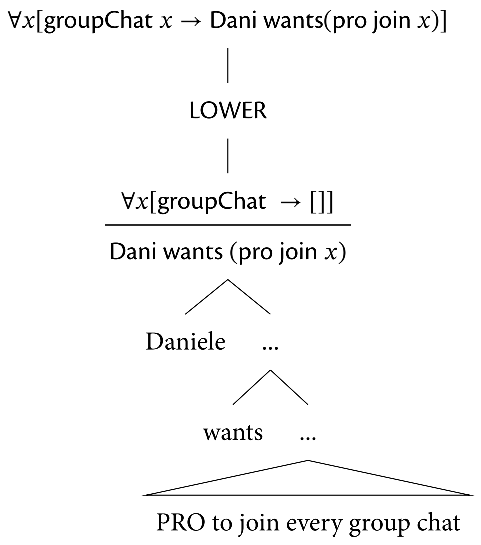
We actually already have everything we need in order to account for this. In a nutshell, the two readings correspond to an application of LOWER either below or above the intensional predicate.[21]

[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. $\text{want} > \forall$


...
Daniele
λy . y wants (∀x[groupChat x → (pro join x)])
wants
∀x[groupChat x → (pro join x)]
LOWER
∀x[groupChat x → []] / pro join x
S
PRO↑
∀x[groupChat x → []] / λy . y join x
S
join↑
∀x[groupChat x → []] / x
every group chat

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


∀x[groupChat x → Dani wants(pro join x)]
LOWER
∀x[groupChat → []] / Dani wants (pro join x)
Daniele
...
wants
...
PRO to join every group chat

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

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

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."				$\exists > \forall$; ✗ $\forall > \exists$
	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. $\exists > \forall$; ✗ $\forall > \exists$
b. Daniele sent a picture to every syntactician. $\exists > \forall$; $\forall > \exists$

bobaljikWurmbrand2012 posit a (violable) economy condition in order to express the preference for surface scope observed in many languages. **bobaljikWurmbrand2012** make architectural assumptions that we aren't necessarily committed to here, but the spirit of **ScoT!** (**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) **ScoT!** (**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*.

6.3 Deriving inverse scope

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

(60) $a^{\uparrow} \coloneqq \frac{[]}{a}$

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.

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

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

Question

What is the *type* of lifted *everyone*?

Going back to tower notation, lifting a tower adds a trivial third story:[22] Following **Charlowc**, 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).

[22] 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.

(63) $\left(\dfrac{\forall x[]}{x}\right)^{\uparrow} = \dfrac{[]}{\dfrac{\forall x[]}{x}}$

Question

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

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

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

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* (⇈).[23]

[23] I can tell what you're thinking: "seriously? Another *darn* type-shifter? How many of these 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.

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

(67) *Internal lift* (def.)

a. $(\upuparrows) : \mathsf{K_t\ a} \to \mathsf{K_t\ (K_t\ a)}$

b. $m^{\upuparrows} := \lambda k\ .\ m\ (\lambda x\ .\ k\ x^{\uparrow})$

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.

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

$$\left(\dfrac{f\,[]}{x}\right)^{\upuparrows} := \dfrac{f\,[]}{\dfrac{[]}{x}}$$

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

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

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

The trick is: we *internally* lift the quantifier that is destined to take wide scope.

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

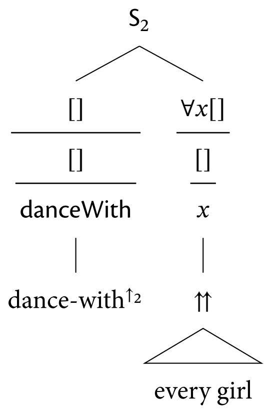
Before we proceed, we need to generalize LIFT and **SFA!** to three-story towers.[24]

(71) $$x^{\uparrow_2} := \begin{array}{c} [] \\ \hline [] \\ \hline x \end{array}$$

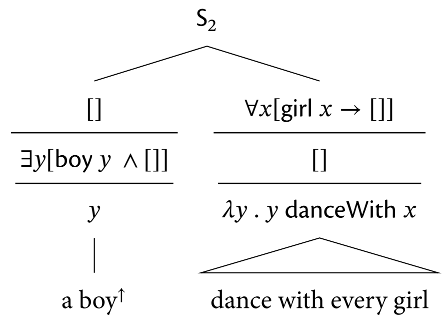
(72) $$\frac{f\,[]}{m}\,\mathsf{S}_2\,\frac{g\,[]}{n} := \frac{f\,(g\,[])}{m\,\mathsf{S}\,n}$$

[24] 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.

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



(74) 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*.[25]

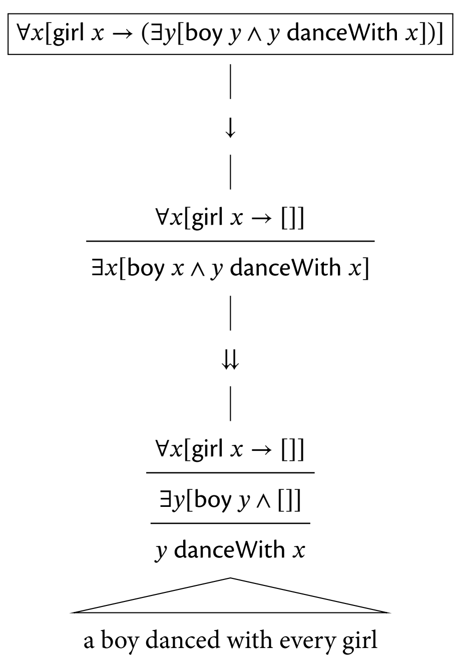
(76) *Internal lower* (def.) $$\left(\begin{array}{c} f\,[] \\ \hline g\,[] \\ \hline p \end{array}\right)^{\downdownarrows} := \begin{array}{c} f\,[] \\ \hline \left(\begin{array}{c} g\,[] \\ \hline x \end{array}\right)^{\downarrow} \end{array}$$

[25] 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:

(75) $m^{\downdownarrows} \equiv (\downarrow)^{\uparrow}\ \mathsf{S}\ m$

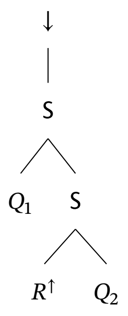
Now we can collapse the tower by doing *internal lower*, followed by *lower*:

(77)

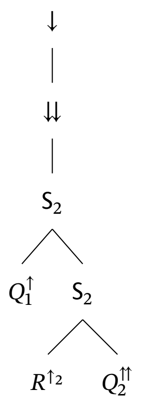

∀x[girl x → (∃y[boy y ∧ y danceWith x])]
↓
∀x[girl x → []]
∃x[boy x ∧ y danceWith x]
⇊
∀x[girl x → []]
∃y[boy y ∧ []]
y danceWith x
a boy danced with every girl

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

(78) Surface scope (schematic derivation)



(79) Inverse scope (schematic derivation)



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, $\uparrow_2$, $\upuparrows$, and $\downdownarrows$.
- In order to derive an inverse scope reading, what was *crucial* was the

availability of *internal lift*; the remaining operations, $\mathsf{S}_2, \uparrow_2, \Downarrow$ 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.[26]

[26] 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*.

If we adopt some version of the *derivational complexity hypothesis*, we also predict that inverse scope readings should take longer to process than surface scope readings. This is something Martin may discuss is a couple of weeks time.

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*:[27]

[27] 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.

(80) *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:

(81) $$\left(\frac{f\,[]}{\dfrac{g\,[]}{x}}\right)^{\mu} = \frac{f\,(g\,[])}{x}$$

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).

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

Arguably, (**??**) 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[28], **Charlowc** develops an interesting take on scope islands couched in terms of continuations.

[28] See, e.g., **danvyFilinski1992** and **wadler1994**.

He proposes the following definition:

(83) *Scope islands* (def.)
A *scope island* is a constituent that is subject to *obligatory evaluation*.
(**Charlowc**)

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 $\mathtt{K_t\ a}$, 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 **Charlowc**, this idea bears an intriguing similarity to **chomskyPhase**'s notion of a *phase*.[29]

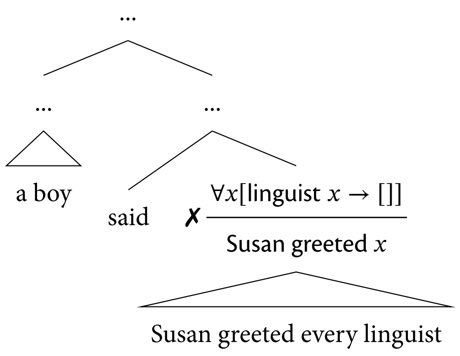
[29] Exploring this parallel in greater depth could make for an interesting term paper topic.

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.

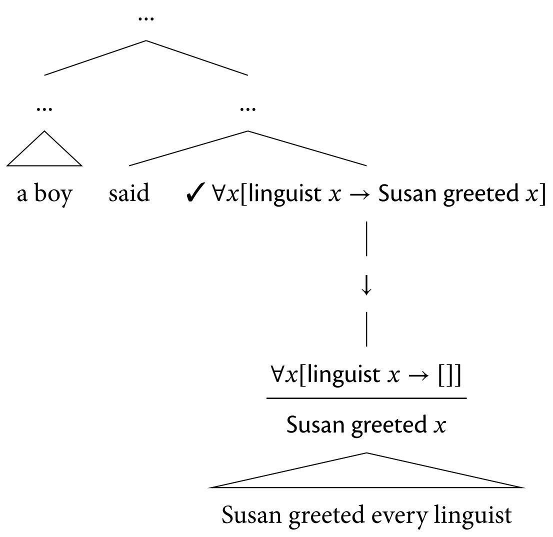
(84) A boy said $\overbrace{\boxed{\text{that Susan greeted every linguist}}}^{\text{scope island}}$. $\exists > \forall; ✗\ \forall > \exists$

The derivation of the embedded clause proceeds as usual via lift and **sfa!**.

(85) Scope island with an unevaluated type



(86) Scope island with an evaluated type


...
...
...
a boy
said
✓ ∀x[linguist x → Susan greeted x]
↓
∀x[linguist x → []]
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?[30]
- 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?[31]

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

[31] 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.

Charlowc 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.[32]

[32] See **demirok2019** for a parallel story couched in more familiar **heimKratzer1998**-esque machinery.

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

8 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:

- 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.

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 **sfa!** (**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., **charlow2018**, **elliott2019applicatives**, 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 $(\mathsf{K}_\mathsf{t}, \uparrow, \mathsf{S})$ is a special case of an *applicative functor*, a highly influential notion in the literature on functional programming (**mcbridePaterson2008**); for applications in linguistic semantics see **kiselyov2017**, **charlow2018**, and **elliott2019applicatives**.

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 $\circledast$.[33] The components of the applicative functor are additionally subject to a number of laws. I give the full definition below:

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

(87) *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`

(88) **Homomorphism**
$f^\eta \circledast x^\eta \equiv (f\ x)^\eta$

(90) **Identity**
$id^\eta \circledast m \equiv m$

(89) **Interchange**
$(\lambda k\ .\ k\ x)^\eta \circledast m \equiv m \circledast x^\eta$

(91) **Composition**
$(\circ)^\eta \circledast u \circledast v \circledast w \equiv u \circledast (v \circledast w)$

You can verify for yourselves that the triple $(\mathtt{K_t}, \uparrow, \mathsf{S})$ 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., **shan2002monads**, **Charlowc**, **grove2019**, 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.

(92) *Monad* (def.)
A *monad* consists of the following three components, subject to **associativity** and **identity**.[34]
a. `F : type → type`
b. η : `a → F a`
c. μ : `F (F a) → F a`

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

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

[34] 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 (F, 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.

We can define join for the continuation monad as follows:

(95) *join* (def.)

a. $\mu : \texttt{K}_\texttt{t}\ (\texttt{K}_\texttt{t}\ \texttt{a}) \to \texttt{K}_\texttt{t}\ \texttt{a}$

b. $m^\mu := \lambda k\ .\ m\ (\lambda c\ .\ c\ k)$

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

(96) $$\frac{f\ []}{\dfrac{\text{g}\ []}{x}}^{\mu} = \frac{f\ (\text{g}\ [])}{x}$$

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:[35]

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

(97) $m^\mu = m \circ (\uparrow)$

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:

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