Continuations and scope

Computational semantics seminar

April 12, 2017

Today

A complete in situ treatment of quantifier scope, using just 3 simple functions (or even 2, depending on how you count).

- Motivating and grokking the basic intuition.
- Exploring a tower notation for making our lives easier.
- Seeing how to talk about scope islands denotationally.

As we go, keep in mind where we've been so far.

- Think about monads, applicatives, functors, and so on.
- Try to see if you notice any commonalities between the approach taken here and our monadic excursions.

Scope

Quantifiers present two basic problems for semantic theory

Quantifiers present two basic problems for semantic theory

Problem 1: how to interpret them in **object positions**?

1. I like everybody.
2. I told a child a scary story.

Quantifiers present two basic problems for semantic theory

Problem 1: how to interpret them in **object positions**?

1. I like everybody.
2. I told a child a scary story.

Problem 2: how to derive **scope ambiguity**?

3. A guard was standing in front of every embassy.
4. A member of each committee voted against Gorsuch.

“Quantifiers”?

Ok, so, probably “quantifiers” is the wrong word, since we are also talking about indefinites, and there are good reasons to think that indefinites might not be quantificational (e.g., exceptional scope and binding).

So it is probably more felicitous to talk about **things that take scope**, and to reframe the problem as follows:

How do things that take scope...

“Quantifiers”?

Ok, so, probably “quantifiers” is the wrong word, since we are also talking about indefinites, and there are good reasons to think that indefinites might not be quantificational (e.g., exceptional scope and binding).

So it is probably more felicitous to talk about **things that take scope**, and to reframe the problem as follows:

How do things that take scope... take scope?

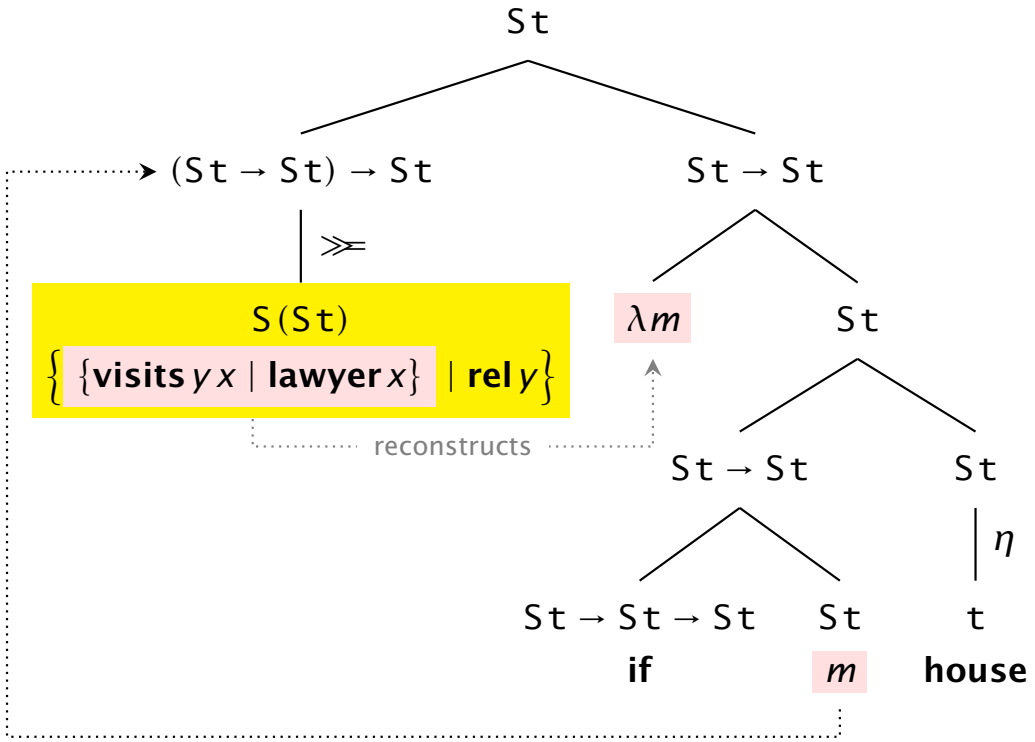
Scope islands

As we're pretty familiar with at this late date, scope-taking is bounded by islands. None of the following sentences readily has a $\forall \gg \exists$ reading.

1. Somebody who [was on every committee] voted against Gorsuch.
2. Someone will be shocked if [every famous linguist is at the party].
3. Somebody thinks that [every linguist is smart].

So maybe the fully general form of the problem is: how do things that take scope take scope **over the things they actually take scope over**?

Zooming out: we've seen derivations like this all semester



$$= \{\textbf{if}\,(\exists x \in \textbf{lawyer} : \textbf{visits}\,y\,x)\,\textbf{house} \mid \textbf{rel}\,y\}$$

In other words

Scope has been central all along: apparent scope out of islands doesn't involve scope out of islands, but it certainly does involve scope!

So far, we have contented ourselves by staying silent about scope-taking. For most cases (though in the end, not all), a bog-standard QR-type mechanism would work just fine (Heim & Kratzer 1998, Büring 2005).

Lexicalism

$$\textbf{saw} :: ((\mathtt{e} \to \mathtt{t}) \to \mathtt{t}) \to \mathtt{e} \to \mathtt{t}$$
$$\textbf{saw} := \lambda\mathcal{F}.\lambda x.\mathcal{F}\,(\lambda y.f\,x\,y)$$

cf. Montague (1974), Muskens (1996), etc

Any problems with this solution?

Lexicalism

$$\textbf{saw} :: ((\mathtt{e} \to \mathtt{t}) \to \mathtt{t}) \to \mathtt{e} \to \mathtt{t}$$
$$\textbf{saw} := \lambda\mathcal{F}.\lambda x.\mathcal{F}\,(\lambda y.f\,x\,y)$$

cf. Montague (1974), Muskens (1996), etc

Any problems with this solution?

- It's not general enough: no inverse scope
- It doesn't handle ditransitive verbs

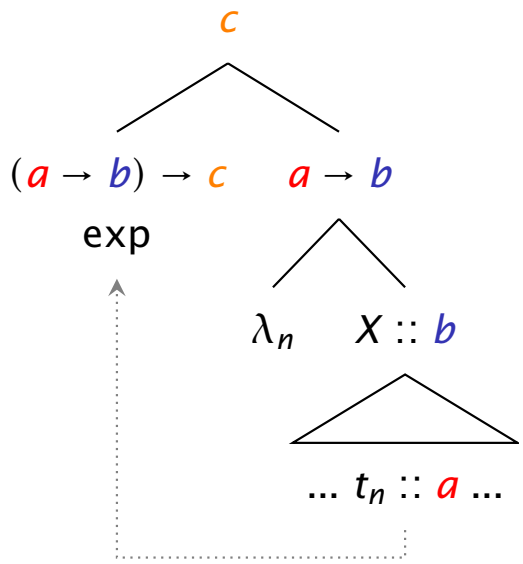
All the scopes!!

We might suppose verbs come in many guises, enough to generate factorial scopings of their arguments. But scope can be quite complex:

1. Reconstruction
2. Comparatives and superlatives
3. “Parasitic” scope
4. Inverse linking

It’s possible to be clever and get your infinite family of operations in a briskly stateable way (Hendriks 1993, cf. also Szabolcsi 2011). Such systems are difficult to use, and the derivations are difficult to construct.

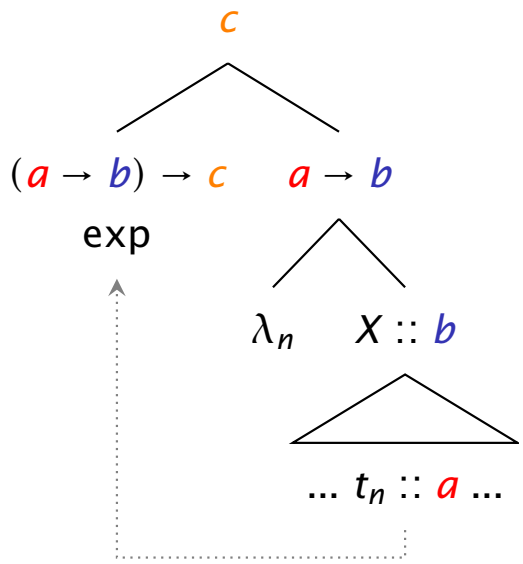
The usual story



Structures like this are interpreted as follows:

$$[\![\text{exp } [\lambda_n\ X]]\!]^g =$$

The usual story



Structures like this are interpreted as follows:

$$[\![\text{exp}\ [\lambda_n\ X]]\!]^g = [\![\text{exp}]\!]\,(\lambda x.[\![X]\!]^{g[n \to x]})$$

In configurations like this, *exp* **scopes over** *X* (and anything inside *X*).

Worries you might have

Is scope-taking really syntactic?

Is quantification really mediated by *assignment functions*? E.g., it's possible to unbind a trace if you're not careful.

Why didn't we pursue an enriched-mode-of-composition approach here, like we did for other cases that got us into trouble?

None of these objections is dispositive, of course.

Abstracting out control

Continuations

A **continuation** is "the rest of a computation":

$$(1 + 3) \times 5$$

Relative to the above computation:

- The continuation of 1 is $\lambda n.\,(n + 3) \times 5$
- The continuation of 3 is $\lambda n.\,(1 + n) \times 5$
- The continuation of 5 is $\lambda n.\,(1 + 3) \times n$

A continuation is the sort of thing you'd get if you QR'd something to the edge of a computation, and then abstracted over its trace.

Clearly, continuations exist independently of any framework or specific analysis, and all occurrences of expressions have continuations in any language that has a semantics. Since continuations are nothing more than a perspective, they are present whether we attend to them or not. The question under consideration, then, is not whether continuations exist — they undoubtedly do — but precisely how natural language expressions do or don't interact with them.

Barker (2002: 215)

Continuization

Once we start attending to continuations, a new possibility opens up: natural language meanings are *functions on their continuations*.

$$(1 + 3) \times 5$$

E.g., $[\![3]\!] = \lambda k.\, k\,3$, $[\![5]\!] = \lambda k.\, k\,5$, $[\![+]\!] = \lambda k.\, k\,(+)$, ...

$$[\![3]\!]\,(\lambda n.\,(1 + n) \times 5) =$$

Continuization

Once we start attending to continuations, a new possibility opens up: natural language meanings are *functions on their continuations*.

$$(1 + 3) \times 5$$

E.g., $[\![3]\!] = \lambda k.\, k\,3$, $[\![5]\!] = \lambda k.\, k\,5$, $[\![+]\!] = \lambda k.\, k\,(+)$, ...

$$[\![3]\!]\,(\lambda n.\,(1 + n) \times 5) = (\lambda k.\, k\,3)\,(\lambda n.\,(1 + n) \times 5)$$
$$=$$

Continuization

Once we start attending to continuations, a new possibility opens up: natural language meanings are *functions on their continuations*.

$$(1 + 3) \times 5$$

E.g., $[\![3]\!] = \lambda k.\, k\,3$, $[\![5]\!] = \lambda k.\, k\,5$, $[\![+]\!] = \lambda k.\, k\,(+)$, ...

$$\begin{aligned} [\![3]\!]\,(\lambda n.\,(1 + n) \times 5) &= (\lambda k.\, k\,3)\,(\lambda n.\,(1 + n) \times 5) \\ &= (\lambda n.\,(1 + n) \times 5)\,3 \\ &= \end{aligned}$$

Continuization

Once we start attending to continuations, a new possibility opens up: natural language meanings are *functions on their continuations*.

$$(1 + 3) \times 5$$

E.g., $[\![3]\!] = \lambda k.\, k\,3$, $[\![5]\!] = \lambda k.\, k\,5$, $[\![+]\!] = \lambda k.\, k\,(+)$, ...

$$\begin{aligned}
[\![3]\!]\,(\lambda n.\,(1 + n) \times 5) &= (\lambda k.\, k\,3)\,(\lambda n.\,(1 + n) \times 5) \\
&= (\lambda n.\,(1 + n) \times 5)\,3 \\
&= (1 + 3) \times 5 \\
&= 20
\end{aligned}$$

Another way to put this: everything does (or at least *can*) **take scope**.

Another arithmetic example, getting more compositional

$$\begin{aligned}[\![\text{sqrt } 2]\!] &= [\![\text{sqrt}]\!]\,(\lambda f.[\![2]\!]\,(\lambda x.f\,x)) \\ &= \end{aligned}$$

Another arithmetic example, getting more compositional

$$
\begin{aligned}
[\![\text{sqrt } 2]\!] &= [\![\text{sqrt}]\!]\,(\lambda f.\,[\![2]\!]\,(\lambda x.\,f\,x)) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,(\lambda k.\,k\,2)\,(\lambda x.\,f\,x)) \\
&=
\end{aligned}
$$

Another arithmetic example, getting more compositional

$$
\begin{aligned}
[\![\text{sqrt } 2]\!] &= [\![\text{sqrt}]\!]\,(\lambda f.\,[\![2]\!]\,(\lambda x.\,f\,x)) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,(\lambda k.\,k\,2)\,(\lambda x.\,f\,x)) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,(\lambda x.\,f\,x)\,2) \\
&=
\end{aligned}
$$

Another arithmetic example, getting more compositional

$$
\begin{aligned}
[\![\text{sqrt } 2]\!] &= [\![\text{sqrt}]\!]\,(\lambda f.\,[\![2]\!]\,(\lambda x.\,f\,x)) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,(\lambda k.\,k\,2)\,(\lambda x.\,f\,x)) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,(\lambda x.\,f\,x)\,2) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,f\,2) \\
&=
\end{aligned}
$$

Another arithmetic example, getting more compositional

$$
\begin{aligned}
[\![\text{sqrt } 2]\!] &= [\![\text{sqrt}]\!]\,(\lambda f.\,[\![2]\!]\,(\lambda x.\,f\,x)) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,(\lambda k.\,k\,2)\,(\lambda x.\,f\,x)) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,(\lambda x.\,f\,x)\,2) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,f\,2) \\
&= (\lambda k.\,k\,\mathbf{sqrt})\,(\lambda f.\,f\,2) \\
&=
\end{aligned}
$$

Another arithmetic example, getting more compositional

$$
\begin{aligned}
[\![\text{sqrt } 2]\!] &= [\![\text{sqrt}]\!]\,(\lambda f.\,[\![2]\!]\,(\lambda x.\,f\,x)) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,(\lambda k.\,k\,2)\,(\lambda x.\,f\,x)) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,(\lambda x.\,f\,x)\,2) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,f\,2) \\
&= (\lambda k.\,k\,\mathbf{sqrt})\,(\lambda f.\,f\,2) \\
&= (\lambda f.\,f\,2)\,\mathbf{sqrt} \\
&=
\end{aligned}
$$

Another arithmetic example, getting more compositional

$$
\begin{aligned}
[\![\text{sqrt } 2]\!] &= [\![\text{sqrt}]\!]\,(\lambda f.\,[\![2]\!]\,(\lambda x.\,f\,x)) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,(\lambda k.\,k\,2)\,(\lambda x.\,f\,x)) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,(\lambda x.\,f\,x)\,2) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,f\,2) \\
&= (\lambda k.\,k\,\mathbf{sqrt})\,(\lambda f.\,f\,2) \\
&= (\lambda f.\,f\,2)\,\mathbf{sqrt} \\
&= \mathbf{sqrt}\,2
\end{aligned}
$$

Artist’s impression

So this solves our problem, right?

$$
\begin{aligned}
[\![\text{sqrt } 2]\!] &= [\![\text{sqrt}]\!]\,(\lambda f.\,[\![2]\!]\,(\lambda x.\,f\,x)) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,(\lambda k.\,k\,2)\,(\lambda x.\,f\,x)) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,(\lambda x.\,f\,x)\,2) \\
&= [\![\text{sqrt}]\!]\,(\lambda f.\,f\,2) \\
&= (\lambda k.\,k\,\mathbf{sqrt})\,(\lambda f.\,f\,2) \\
&= (\lambda f.\,f\,2)\,\mathbf{sqrt} \\
&= \mathbf{sqrt}\,2
\end{aligned}
$$

So this solves our problem, right?

$$
\begin{aligned}
[\![\text{saw everyone}]\!] &= [\![\text{saw}]\!](\lambda f.[\![\text{everyone}]\!](\lambda x.f\,x)) \\
&= [\![\text{saw}]\!](\lambda f.(\lambda k.\forall y.k\,y)\,(\lambda x.f\,x)) \\
&= [\![\text{saw}]\!](\lambda f.\forall y.(\lambda x.f\,x)\,y) \\
&= [\![\text{saw}]\!](\lambda f.\forall y.f\,y) \\
&= (\lambda k.k\,\mathbf{saw})\,(\lambda f.\forall y.f\,y) \\
&= (\lambda f.\forall y.f\,y)\,\mathbf{saw} \\
&= \forall y.\mathbf{saw}\,y
\end{aligned}
$$

So this solves our problem, right?

$$
\begin{aligned}
[\![\text{saw everyone}]\!] &= [\![\text{saw}]\!]\,(\lambda f.\,[\![\text{everyone}]\!]\,(\lambda x.\,f\,x)) \\
&= [\![\text{saw}]\!]\,(\lambda f.\,(\lambda k.\,\forall y.\,k\,y)\,(\lambda x.\,f\,x)) \\
&= [\![\text{saw}]\!]\,(\lambda f.\,\forall y.\,(\lambda x.\,f\,x)\,y) \\
&= [\![\text{saw}]\!]\,(\lambda f.\,\forall y.\,f\,y) \\
&= (\lambda k.\,k\,\mathbf{saw})\,(\lambda f.\,\forall y.\,f\,y) \\
&= (\lambda f.\,\forall y.\,f\,y)\mathbf{saw} \\
&= \forall y.\,\mathbf{saw}\,y \quad \#\#\#
\end{aligned}
$$

The result is not well typed! It's just like we attempted to apply $[\![\text{everyone}]\!]$ to $[\![\text{saw}]\!]$ directly!!

Asymmetry

Do you see what went wrong? Do you have any ideas how to fix it?

Asymmetry

Do you see what went wrong? Do you have any ideas how to fix it?

My take: we were *unsystematic* in continuizing the grammar. If you have two scopal meanings, and you combine them by scoping them, the result of that should be. . .

Asymmetry

Do you see what went wrong? Do you have any ideas how to fix it?

My take: we were *unsystematic* in continuizing the grammar. If you have two scopal meanings, and you combine them by scoping them, the result of that should be. . . **another scopal expression**!

$$\lambda k.\, \forall\, y.\, k\,(\mathbf{saw}\, y)$$

What type does k need to have for this to be well-typed?

Asymmetry

Do you see what went wrong? Do you have any ideas how to fix it?

My take: we were *unsystematic* in continuizing the grammar. If you have two scopal meanings, and you combine them by scoping them, the result of that should be... **another scopal expression**!

$$\lambda k.\, \forall y.\, k\,(\mathbf{saw}\, y)$$

What type does k need to have for this to be well-typed?

$$(\mathtt{e} \to \mathtt{t}) \to \mathtt{t}$$

So the type of *saw everyone* is $((\mathtt{e} \to \mathtt{t}) \to \mathtt{t}) \to \mathtt{t}$! A scopal VP!

Continuized functional application

We can factor out the necessary operation:

$$F /\!\!/ X := \lambda k.\, F\,(\lambda f.\, X\,(\lambda x.\, k\,(f\,x)))$$

What type does the $/\!\!/$ function have?

Continuized functional application

We can factor out the necessary operation:

$$F /\!\!/ X := \lambda k.\, F\,(\lambda f.\, X\,(\lambda x.\, k\,(f\,x)))$$

What type does the $/\!\!/$ function have?

$$(((a \to b) \to \mathtt{t}) \to \mathtt{t}) \to ((a \to \mathtt{t}) \to \mathtt{t}) \to (b \to \mathtt{t}) \to \mathtt{t}$$

[Actually, its type is considerably more general than this! But this artificially restricted type will get us pretty far today.]

Fixing the asymmetry

Similarly, there is no need to posit (as Montague did) that the *lexical meaning* of *saw* lives in this higher type. We can systematically derive such meanings on the fly via an auxiliary function. What is it?

$$x^{\uparrow} := \lambda k.\, k\, x$$

aka the **LIFT** shift (Partee 1986)

What type does the ↑ function have?

Fixing the asymmetry

Similarly, there is no need to posit (as Montague did) that the *lexical meaning* of *saw* lives in this higher type. We can systematically derive such meanings on the fly via an auxiliary function. What is it?

$$x^{\uparrow} := \lambda k.\, k\, x$$

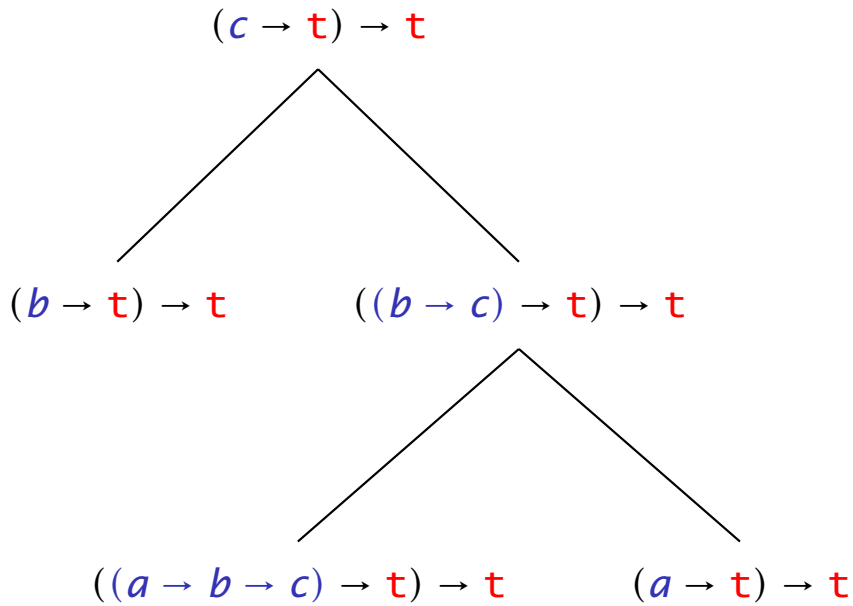
aka the **LIFT** shift (Partee 1986)

What type does the ↑ function have?

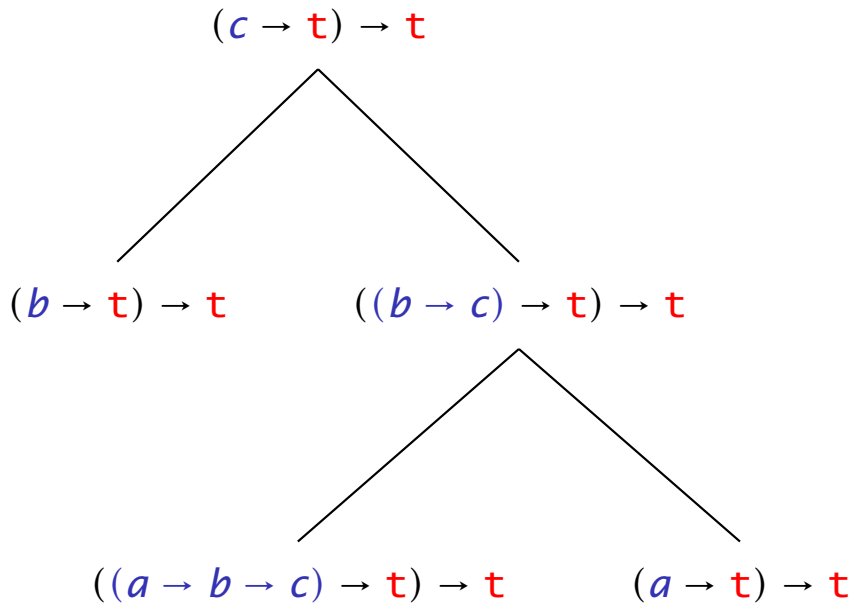
$$a \to (a \to \mathtt{t}) \to \mathtt{t}$$

[Again, its type is in fact considerably more general than this! But again, this artificially restricted type will get us pretty far today.]

In tree form


$(c \to \mathtt{t}) \to \mathtt{t}$
$(b \to \mathtt{t}) \to \mathtt{t}$
$((b \to c) \to \mathtt{t}) \to \mathtt{t}$
$((a \to b \to c) \to \mathtt{t}) \to \mathtt{t}$
$(a \to \mathtt{t}) \to \mathtt{t}$

In tree form



It's as if we do functional application on the expressions' "core" values, while balancing their scopal effects on our heads.

Deriving *Mary left*, continuations-style

$$\lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\mathbf{m}^{\uparrow}\,(\lambda x.\,k\,(f\,x)))\;=$$

Deriving *Mary left*, continuations-style

$$\lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\mathbf{m}^{\uparrow}\,(\lambda x.\,k\,(f\,x))) = \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda k'.\,k'\,\mathbf{m})\,(\lambda x.\,k\,(f\,x)))$$
$$=$$

Deriving *Mary left*, continuations-style

$$\begin{aligned}
\lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\mathbf{m}^{\uparrow}\,(\lambda x.\,k\,(f\,x))) &= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda k'.\,k'\,\mathbf{m})\,(\lambda x.\,k\,(f\,x))) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda x.\,k\,(f\,x))\,\mathbf{m}) \\
&=
\end{aligned}$$

Deriving *Mary left*, continuations-style

$$\begin{aligned}
\lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\mathbf{m}^{\uparrow}\,(\lambda x.\,k\,(f\,x))) &= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda k'.\,k'\,\mathbf{m})\,(\lambda x.\,k\,(f\,x))) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda x.\,k\,(f\,x))\,\mathbf{m}) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,k\,(f\,\mathbf{m})) \\
&=
\end{aligned}$$

Deriving *Mary left*, continuations-style

$$
\begin{aligned}
\lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\mathbf{m}^{\uparrow}\,(\lambda x.\,k\,(f\,x))) &= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda k'.\,k'\,\mathbf{m})\,(\lambda x.\,k\,(f\,x))) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda x.\,k\,(f\,x))\,\mathbf{m}) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,k\,(f\,\mathbf{m})) \\
&= \lambda k.\,(\lambda k'.\,k'\,\mathbf{left})\,(\lambda f.\,k\,(f\,\mathbf{m})) \\
&=
\end{aligned}
$$

Deriving *Mary left*, continuations-style

$$
\begin{aligned}
\lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\mathbf{m}^{\uparrow}\,(\lambda x.\,k\,(f\,x))) &= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda k'.\,k'\,\mathbf{m})\,(\lambda x.\,k\,(f\,x))) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda x.\,k\,(f\,x))\,\mathbf{m}) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,k\,(f\,\mathbf{m})) \\
&= \lambda k.\,(\lambda k'.\,k'\,\mathbf{left})\,(\lambda f.\,k\,(f\,\mathbf{m})) \\
&= \lambda k.\,(\lambda f.\,k\,(f\,\mathbf{m}))\,\mathbf{left} \\
&=
\end{aligned}
$$

Deriving *Mary left*, continuations-style

$$
\begin{aligned}
\lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\mathbf{m}^{\uparrow}\,(\lambda x.\,k\,(f\,x))) &= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda k'.\,k'\,\mathbf{m})\,(\lambda x.\,k\,(f\,x))) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda x.\,k\,(f\,x))\,\mathbf{m}) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,k\,(f\,\mathbf{m})) \\
&= \lambda k.\,(\lambda k'.\,k'\,\mathbf{left})\,(\lambda f.\,k\,(f\,\mathbf{m})) \\
&= \lambda k.\,(\lambda f.\,k\,(f\,\mathbf{m}))\,\mathbf{left} \\
&= \lambda k.\,k\,(\mathbf{left}\,\mathbf{m}) \\
&=
\end{aligned}
$$

Deriving *Mary left*, continuations-style

$$
\begin{aligned}
\lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\mathbf{m}^{\uparrow}\,(\lambda x.\,k\,(f\,x))) &= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda k'.\,k'\,\mathbf{m})\,(\lambda x.\,k\,(f\,x))) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda x.\,k\,(f\,x))\,\mathbf{m}) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,k\,(f\,\mathbf{m})) \\
&= \lambda k.\,(\lambda k'.\,k'\,\mathbf{left})\,(\lambda f.\,k\,(f\,\mathbf{m})) \\
&= \lambda k.\,(\lambda f.\,k\,(f\,\mathbf{m}))\,\mathbf{left} \\
&= \lambda k.\,k\,(\mathbf{left}\,\mathbf{m}) \\
&= (\mathbf{left}\,\mathbf{m})^{\uparrow}
\end{aligned}
$$

More generally, we have the following:

$$f^{\uparrow} \mathbin{/\!\!/} x^{\uparrow} = (f\,x)^{\uparrow}$$

The ↑ intuition

Given these remarks, what should a continuized derivation of a simple transitive sentence (no quantifiers) look like?

We could go through an involved, tedious series of β-reductions, but there is honestly no need:

The ↑ intuition

Given these remarks, what should a continuized derivation of a simple transitive sentence (no quantifiers) look like?

We could go through an involved, tedious series of β-reductions, but there is honestly no need:

$$\rightsquigarrow (R\,y\,x)^{\uparrow} = \lambda k.\, k\,(R\,y\,x)$$

And the same goes for ditransitives, possessive nouns, etc etc.

Deriving *everyone left*, continuations-style

$$\lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\mathbf{eo}(\lambda x.\,k\,(f\,x)))\;=$$

Deriving *everyone left*, continuations-style

$$\lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\mathbf{eo}\,(\lambda x.\,k\,(f\,x))) = \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda k'.\,\forall y.\,k'\,y)\,(\lambda x.\,k\,(f\,x)))$$
$$=$$

Deriving *everyone left*, continuations-style

$$\begin{aligned}
\lambda k.\,\mathbf{left}^{\uparrow}(\lambda f.\,\mathbf{eo}(\lambda x.\,k\,(f\,x))) &= \lambda k.\,\mathbf{left}^{\uparrow}(\lambda f.\,(\lambda k'.\,\forall y.\,k'\,y)\,(\lambda x.\,k\,(f\,x))) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}(\lambda f.\,\forall y.\,(\lambda x.\,k\,(f\,x))\,y) \\
&=
\end{aligned}$$

Deriving *everyone left*, continuations-style

$$
\begin{aligned}
\lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\mathbf{eo}\,(\lambda x.\,k\,(f\,x))) &= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda k'.\,\forall y.\,k'\,y)\,(\lambda x.\,k\,(f\,x))) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\forall y.\,(\lambda x.\,k\,(f\,x))\,y) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\forall y.\,k\,(f\,y)) \\
&=
\end{aligned}
$$

Deriving *everyone left*, continuations-style

$$
\begin{aligned}
\lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\mathbf{eo}\,(\lambda x.\,k\,(f\,x))) &= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,(\lambda k'.\,\forall y.\,k'\,y)\,(\lambda x.\,k\,(f\,x))) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\forall y.\,(\lambda x.\,k\,(f\,x))\,y) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}\,(\lambda f.\,\forall y.\,k\,(f\,y)) \\
&= \lambda k.\,(\lambda k'.\,k'\,\mathbf{left})\,(\lambda f.\,\forall y.\,k\,(f\,y)) \\
&=
\end{aligned}
$$

Deriving *everyone left*, continuations-style

$$
\begin{aligned}
\lambda k.\,\mathbf{left}^{\uparrow}(\lambda f.\,\mathbf{eo}(\lambda x.\,k\,(f\,x))) &= \lambda k.\,\mathbf{left}^{\uparrow}(\lambda f.\,(\lambda k'.\,\forall y.\,k'\,y)\,(\lambda x.\,k\,(f\,x))) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}(\lambda f.\,\forall y.\,(\lambda x.\,k\,(f\,x))\,y) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}(\lambda f.\,\forall y.\,k\,(f\,y)) \\
&= \lambda k.\,(\lambda k'.\,k'\,\mathbf{left})\,(\lambda f.\,\forall y.\,k\,(f\,y)) \\
&= \lambda k.\,(\lambda f.\,\forall y.\,k\,(f\,y))\,\mathbf{left} \\
&=
\end{aligned}
$$

Deriving *everyone left*, continuations-style

$$
\begin{aligned}
\lambda k.\,\mathbf{left}^{\uparrow}(\lambda f.\,\mathbf{eo}(\lambda x.\,k\,(f\,x))) &= \lambda k.\,\mathbf{left}^{\uparrow}(\lambda f.\,(\lambda k'.\,\forall y.\,k'\,y)\,(\lambda x.\,k\,(f\,x))) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}(\lambda f.\,\forall y.\,(\lambda x.\,k\,(f\,x))\,y) \\
&= \lambda k.\,\mathbf{left}^{\uparrow}(\lambda f.\,\forall y.\,k\,(f\,y)) \\
&= \lambda k.\,(\lambda k'.\,k'\,\mathbf{left})\,(\lambda f.\,\forall y.\,k\,(f\,y)) \\
&= \lambda k.\,(\lambda f.\,\forall y.\,k\,(f\,y))\,\mathbf{left} \\
&= \lambda k.\,\forall y.\,k\,(\mathbf{left}\,y)
\end{aligned}
$$

As in the previous derivation, the verb meaning made its way into the argument of k, but part of the subject stayed outside k. Which part?

- The $\forall y$, which **began its life** outside k: $\lambda k.\,\forall y.\,k\,y$!

The // intuition

So that's how continuized functional application works: the "core" values filter down until they're inside k. The interesting, scopal bits of meaning, remain outside k:

$$(\lambda k.A[kf]) \mathbin{//} (\lambda k.B[kx]) =$$

The // intuition

So that's how continuized functional application works: the "core" values filter down until they're inside k. The interesting, scopal bits of meaning, remain outside k:

$$(\lambda k.A[kf]) \mathbin{//} (\lambda k.B[kx]) = \lambda k.A[B[k(fx)]]$$

This makes it easy to construct and reason about continuized derivations, despite the huge number of β-reductions that they imply.

Deriving *someone saw everyone*

We won't run through this one in detail. But probably you already have enough of the intuition in place to see how it will go.

First, we derive a meaning for the VP:

$$\mathbf{saw}^{\uparrow} /\!\!/ \mathbf{eo} =$$

Deriving *someone saw everyone*

We won't run through this one in detail. But probably you already have enough of the intuition in place to see how it will go.

First, we derive a meaning for the VP:

$$\mathbf{saw}^{\uparrow} \mathbin{/\!\!/} \mathbf{eo} = \lambda k.\, \forall y.\, k\,(\mathbf{saw}\, y)$$

Then, folding in the subject:

$$(\mathbf{saw}^{\uparrow} \mathbin{/\!\!/} \mathbf{eo}) \mathbin{/\!\!/} \mathbf{so} =$$

Deriving *someone saw everyone*

We won't run through this one in detail. But probably you already have enough of the intuition in place to see how it will go.

First, we derive a meaning for the VP:

$$\mathbf{saw}^{\uparrow} \mathbin{/\!\!/} \mathbf{eo} = \lambda k.\, \forall y.\, k\,(\mathbf{saw}\, y)$$

Then, folding in the subject:

$$(\mathbf{saw}^{\uparrow} \mathbin{/\!\!/} \mathbf{eo}) \mathbin{/\!\!/} \mathbf{so} = \lambda k.\, \forall y.\, \exists x.\, k\,(\mathbf{saw}\, y\, x)$$

This is... the inverse scope derivation! Why?

Surface scope?

As you hopefully figured out we only derive inverse scope because $/\!\!/$ gives the "function-y" thing scope over the "argument-y" thing.

One way to get around this would be to admit a second $/\!\!/$ rule:

$$F \,/\!\!/' \, X := \lambda k.X\,(\lambda x.F\,(\lambda f.k\,(f\,x)))$$

$$F \,/\!\!/ \;\; X := \lambda k.F\,(\lambda f.X\,(\lambda x.k\,(f\,x)))$$

Can you think of arguments for or against this approach?

Against $/\!/'$?

Sentences like the following have a reading that $/\!/$ and $/\!/'$ cannot derive:

1. Two people sent a letter to every student.

Which reading do you suppose that is?

Against $/\!/'$?

Sentences like the following have a reading that $/\!/$ and $/\!/'$ cannot derive:

1. Two people sent a letter to every student.

Which reading do you suppose that is?

Yep, $\forall \gg 2 \gg \exists$ is a possible reading of the sentence, but $/\!/$ and $/\!/'$ can't generate it. Let's focus on how the VP and subject compose:

- If $/\!/$ is used, $\forall$ and $\exists$ will **both** scope over 2
- If $/\!/'$ is used, 2 will scope over **both** $\forall$ and $\exists$

We will circle back to this point in the next section.

A sad vignette

1. (What was the party like?)
   Oh it was awful. Nobody came, and I had to clean up!

What meaning does $/\!/$ derive for *nobody came*?

A sad vignette

1. (What was the party like?)
   Oh it was awful. Nobody came, and I had to clean up!

What meaning does $/\!/$ derive for *nobody came*?

$$\lambda k.\, \neg \exists x.\, k\,(\mathbf{came}\, x)$$

It's still raring to go! If we keep composing the sentence, we get:

A sad vignette

1. (What was the party like?)
   Oh it was awful. Nobody came, and I had to clean up!

What meaning does $/\!/$ derive for *nobody came*?

$$\lambda k.\neg\exists x.\, k\,(\mathbf{came}\, x)$$

It's still raring to go! If we keep composing the sentence, we get:

$$\lambda k.\neg\exists x.\, k\,(\mathbf{came}\, x \wedge \mathbf{i.cleaned})$$

Is this... ok?

A sad vignette

1. (What was the party like?)
   Oh it was awful. Nobody came, and I had to clean up!

What meaning does $/\!/$ derive for *nobody came*?

$$\lambda k.\, \neg \exists x.\, k\,(\mathbf{came}\, x)$$

It's still raring to go! If we keep composing the sentence, we get:

$$\lambda k.\, \neg \exists x.\, k\,(\mathbf{came}\, x \wedge \mathbf{i.cleaned})$$

Is this... ok? *Hell* no! It's true if I didn't clean!

Enforcing islands

This suggests that we need a way to end derivations, and that derivations must be obligatorily concluded at certain points (e.g., at tensed clauses).

Can you think of a way to "conclude" $\lambda k.\neg\exists x.k\,(\mathbf{came}\,x)$?

Enforcing islands

This suggests that we need a way to end derivations, and that derivations must be obligatorily concluded at certain points (e.g., at tensed clauses).

Can you think of a way to "conclude" $\lambda k.\neg\exists x.k\,(\mathbf{came}\,x)$?

Sure, turn it into something of type $\mathtt{t}$!! How?

Enforcing islands

This suggests that we need a way to end derivations, and that derivations must be obligatorily concluded at certain points (e.g., at tensed clauses).

Can you think of a way to "conclude" $\lambda k.\neg\exists x.k\,(\mathbf{came}\,x)$?

Sure, turn it into something of type $\mathtt{t}!!$ How?

$$(\lambda k.\neg\exists x.k\,(\mathbf{came}\,x))\,(\lambda p.p) = \neg\exists x.\mathbf{came}\,x$$

If we re-$\uparrow$ this, we complete something known to computer scientists as a **reset** (cf. Barker 2002): $\lambda k.k\,(\neg\exists x.\mathbf{came}\,x)$.

Islands, enforced

$\lambda k.\, k\,(\neg \exists x.\, \textbf{came}\, x)$ is quite a different beast from $\lambda k.\, \neg \exists x.\, k\,(\textbf{came}\, x)$.

The former is done taking (non-trivial) scope. The latter is still spoiling for some scope-taking.

Together, these facts suggest that we must lower (or reset) at clause boundaries, on pain of massive over-generation (compare this to the usual prohibition on QR out of a tensed clause).

Towers

The tower notation

Continuized derivations are easier to appreciate if we help outselves to an ingenious bit of notation known as **tower** (Barker & Shan 2008):

$$\lambda k.f\,[k\,x] \rightsquigarrow \frac{f\,[\ ]}{x}$$

A few examples of how this works:

$\lambda k.k\,\mathbf{m} \rightsquigarrow$

The tower notation

Continuized derivations are easier to appreciate if we help outselves to an ingenious bit of notation known as **tower** (Barker & Shan 2008):

$$\lambda k.\, f\,[k\,x] \rightsquigarrow \begin{array}{c} f\,[\,] \\ \hline x \end{array}$$

A few examples of how this works:

$$\lambda k.\, k\,\mathbf{m} \rightsquigarrow \begin{array}{c} [\,] \\ \hline \mathbf{m} \end{array} \qquad \lambda k.\, k\,\mathbf{saw} \rightsquigarrow$$

The tower notation

Continuized derivations are easier to appreciate if we help outselves to an ingenious bit of notation known as **tower** (Barker & Shan 2008):

$$\lambda k.f\,[k\,x] \rightsquigarrow \frac{f\,[\,]}{x}$$

A few examples of how this works:

$$\lambda k.k\,\mathbf{m} \rightsquigarrow \frac{[\,]}{\mathbf{m}} \qquad \lambda k.k\,\mathbf{saw} \rightsquigarrow \frac{[\,]}{\mathbf{saw}} \qquad \lambda k.\forall y.k\,y \rightsquigarrow$$

The tower notation

Continuized derivations are easier to appreciate if we help outselves to an ingenious bit of notation known as **tower** (Barker & Shan 2008):

$$\lambda k.\, f\,[k\,x] \rightsquigarrow \frac{f\,[\,]}{x}$$

A few examples of how this works:

$$\lambda k.\, k\,\mathbf{m} \rightsquigarrow \frac{[\,]}{\mathbf{m}} \qquad \lambda k.\, k\,\mathbf{saw} \rightsquigarrow \frac{[\,]}{\mathbf{saw}} \qquad \lambda k.\, \forall y.\, k\,y \rightsquigarrow \frac{\forall y.\,[\,]}{y}$$

In other words, towers give a way to visually separate the inherently scopal parts of meaning from the function-argument backbone.

Tower combination

Recall our intuition about how continuized combination works:

$$(\lambda k.A[kf]) /\!\!/ (\lambda k.B[kx]) = \lambda k.A[B[k(fx)]]$$

This can be naturally re-expressed in the tower notation:

$$\frac{A[\ ]}{f}\ \frac{B[\ ]}{x} \rightsquigarrow \frac{A[B[\ ]]}{fx}$$

Example derivation

$$\frac{\exists x.[\ ]}{x}\left(\frac{[\ ]}{\textbf{saw}}\ \frac{\forall y.[\ ]}{y}\right) \rightsquigarrow$$

Example derivation

$$\frac{\exists x.[\ ]}{x}\left(\frac{[\ ]}{\textbf{saw}}\ \frac{\forall y.[\ ]}{y}\right) \rightsquigarrow \frac{\exists x.\forall y.[\ ]}{\textbf{saw}\,y\,x}$$

Notice that I'm assuming (for simplicity) that the thing to the left always scopes over the thing to the right. In other words, we might define a general mode of combination, as follows:

Example derivation

$$\frac{\exists x.[\ ]}{x}\left(\frac{[\ ]}{\textbf{saw}}\ \frac{\forall y.[\ ]}{y}\right) \rightsquigarrow \frac{\exists x.\forall y.[\ ]}{\textbf{saw}\,y\,x}$$

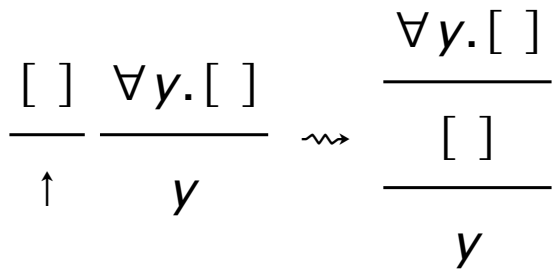
Notice that I'm assuming (for simplicity) that the thing to the left always scopes over the thing to the right. In other words, we might define a general mode of combination, as follows:

$$X \parallel Y := \lambda k.X\,(\lambda x.Y\,(\lambda y.\textbf{Combine}\,(x,y)))$$

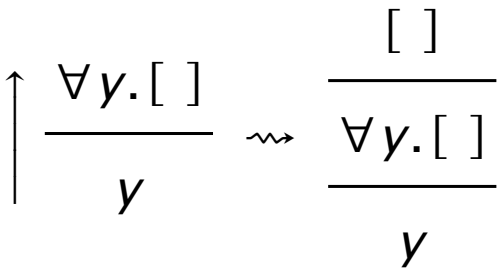
In other words, we do forwards or backwards functional application on the "bottom" story, depending on the types of the expressions involved.

Varieties of lift

Notice that we can actually apply operations *inside* towers:



As well as to towers:



A note on **Combine**

Because ↑ (i.e., **LIFT**) inverts function-argument relationships, we can actually make do with ⫽ alone!

$$\left(\frac{[\,]}{\uparrow}\ \frac{\forall y.[\,]}{y}\right)\frac{[\,]}{\mathbf{left}} \rightsquigarrow \frac{\forall y.[\,]}{\lambda k.\,k\,y}\ \frac{[\,]}{\mathbf{left}} \rightsquigarrow \frac{\forall y.[\,]}{\mathbf{left}\,y}$$

This derivation is composed using nothing other than ⫽.

Big tower combination

$$\frac{\dfrac{A[\,]}{C[\,]}\quad\dfrac{B[\,]}{D[\,]}}{\dfrac{}{f}\quad\dfrac{}{x}} \rightsquigarrow \frac{\dfrac{A[B[\,]]}{C[D[\,]]}}{f\,x}$$

In fact, this rule follows directly from applying $/\!/$ *inside* the function tower. There is no need to stipulate it separately:

$$\left(\frac{[\,]\quad\dfrac{A[\,]}{C[\,]}}{/\!/\quad\dfrac{}{f}}\right)\frac{\dfrac{B[\,]}{D[\,]}}{x} = \frac{\dfrac{A[B[\,]]}{C[D[\,]]}}{f\,x}$$

And the same goes for towers of arbitary height.

Inverse scope

And that is all we need to account for inverse scope!

$$\frac{[\,]}{\dfrac{\exists x.[\,]}{x}}\left(\frac{[\,]}{\dfrac{[\,]}{\mathbf{saw}}}\;\frac{\forall y.[\,]}{[\,]}\right) \rightsquigarrow \frac{\forall y.[\,]}{\dfrac{\exists x.[\,]}{\mathbf{saw}\,y\,x}}$$

Lowering

The last piece is re-casting lowering in terms of towers. Like combination, it works automatically for towers of arbitary heights.

$$\frac{f[\,]}{x} \rightsquigarrow f[x] \qquad \frac{\dfrac{f[\,]}{g[\,]}}{x} \rightsquigarrow f[g[x]]$$

Lowering our inverse-scope derivation:

$$\frac{\dfrac{\forall y.[\,]}{\exists x.[\,]}}{\mathbf{saw}\,y\,x} \rightsquigarrow \forall y.\exists x.\mathbf{saw}\,y\,x$$

A note on lowering

Lowering requires us to conjure an identity function. Along with $\uparrow$ and $/\!/$, that makes three functions appealed to.

Notice that the identity function is η-equivalent to function application:

$$\lambda f.\lambda x.f\,x =_\eta \lambda f.f$$

So from a certain point of view, the only real additions required to use continuations are $\uparrow$ and $/\!/$!

- And remember that if we're using monads, $\uparrow$ comes for free! One of the monad laws has it that $(\eta\,x)^{\gg=} = \lambda k.k\,x = x^{\uparrow}$.
- We'll tug on this very interesting thread more next time.

References I

Barker, Chris. 2002. Continuations and the nature of quantification. *Natural Language Semantics* 10(3). 211–242. http://dx.doi.org/10.1023/A:1022183511876.

Barker, Chris & Chung-chieh Shan. 2008. Donkey anaphora is in-scope binding. *Semantics and Pragmatics* 1(1). 1–46. http://dx.doi.org/10.3765/sp.1.1.

Büring, Daniel. 2005. *Binding theory*. New York: Cambridge University Press. http://dx.doi.org/10.1017/cbo9780511802669.

Heim, Irene & Angelika Kratzer. 1998. *Semantics in generative grammar*. Oxford: Blackwell.

Hendriks, Herman. 1993. *Studied flexibility: Categories and types in syntax and semantics*. University of Amsterdam Ph.D. thesis.

Montague, Richard. 1974. The proper treatment of quantification in ordinary English. In Richmond Thomason (ed.), *Formal Philosophy*, chap. 8, 247–270. New Haven: Yale University Press.

Muskens, Reinhard. 1996. Combining Montague semantics and discourse representation. *Linguistics and Philosophy* 19(2). 143–186. http://dx.doi.org/10.1007/BF00635836.

Partee, Barbara H. 1986. Noun phrase interpretation and type-shifting principles. In Jeroen Groenendijk, Dick de Jongh & Martin Stokhof (eds.), *Studies in Discourse Representation Theory and the Theory of Generalized Quantifiers*, 115–143. Dordrecht: Foris.

References II

Szabolcsi, Anna. 2011. Scope and binding. In Klaus von Heusinger, Claudia Maienborn & Paul Portner (eds.), *Semantics: An international handbook of natural language meaning*, vol. 33 (HSK 2), chap. 62, 1605–1641. Berlin: de Gruyter. http://dx.doi.org/10.1515/9783110255072.1605.