In situ scope-taking

Semantics II

April 23 & 25, 2018

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.

Applicatives (McBride & Paterson 2008, Kiselyov 2015)

A type constructor F is applicative if it supports ρ and $\circledast$ with these types...

$$\rho : a \to F\,a \qquad\qquad \circledast : F\,(a \to b) \to F\,a \to F\,b$$

...Where ρ is a trivial way to inject something into the richer type characterized by F, and $\circledast$ is function application lifted into F.[1]

Applicatives can be pulled more or less directly out of standard approaches to assignment-dependence and alternative semantics.

[1] To ensure that ρ and $\circledast$ behave as advertised, they'll need to satisfy some laws. These needn't detain us, but see McBride & Paterson 2008, Charlow 2017.

The assignment-dependence applicative

We start by characterizing the relevant notion of fanciness:

$$\mathsf{G}\,a ::= \mathsf{g} \to a$$

Then we look for ρ and $\circledast$ with the right types:

$$\underbrace{\rho\, x := \lambda g.\, x}_{\text{cf. } [\![\text{John}]\!] := \lambda g.\mathbf{j}} \qquad\qquad \underbrace{m \circledast n := \lambda g.\, m g\, (n g)}_{\text{cf. } [\![\alpha\ \beta]\!] := \lambda g.[\![\alpha]\!]\, g\, ([\![\beta]\!]\, g)}$$

The alternatives applicative

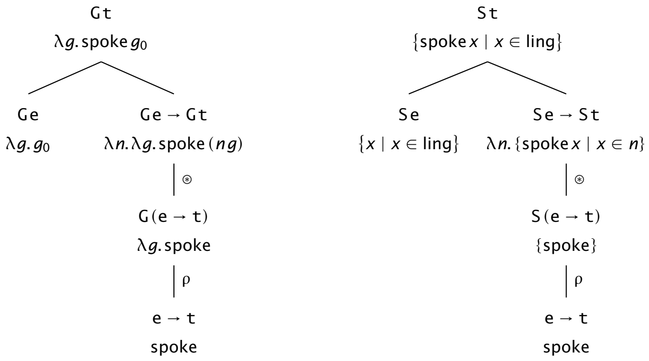
The technique is quite general. Alternatives follow a similar pattern:

$$\mathsf{S}\,a ::= a \to \{0, 1\}$$

Then S's ρ and $\circledast$ operations are defined as follows:

$$\underbrace{\rho\, x := \{x\}}_{\text{cf. } [\![\text{John}]\!] := \{\text{j}\}} \qquad\qquad \underbrace{m \circledast n := \{f\,x \mid f \in m, x \in n\}}_{\text{cf. } [\![\alpha\ \beta]\!] := \{f\,x \mid f \in [\![\alpha]\!], x \in [\![\beta]\!]\}}$$

Sample derivations: *she$_0$ spoke/a linguist spoke*



$\mathtt{G\,t}$
$\lambda g.\,\text{spoke}\,g_0$
$\mathtt{G\,e}$
$\lambda g.\,g_0$
$\mathtt{G\,e} \to \mathtt{G\,t}$
$\lambda n.\lambda g.\,\text{spoke}\,(n\,g)$
$\circledast$
$\mathtt{G\,(e \to t)}$
$\lambda g.\,\text{spoke}$
ρ
$\mathtt{e \to t}$
spoke
$\mathtt{S\,t}$
$\{\text{spoke}\,x \mid x \in \text{ling}\}$
$\mathtt{S\,e}$
$\{x \mid x \in \text{ling}\}$
$\mathtt{S\,e} \to \mathtt{S\,t}$
$\lambda n.\,\{\text{spoke}\,x \mid x \in n\}$
$\circledast$
$\mathtt{S\,(e \to t)}$
$\{\text{spoke}\}$
ρ
$\mathtt{e \to t}$
spoke

Scope

Quantifiers present two basic problems for semantic theory

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

1. I like everybody.
2. I told every 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.

Scope islands

Scope-taking is bounded by scope islands. None of these 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**?

Lexicalism

$$[\![\text{saw}]\!] = \underbrace{\lambda X.\lambda y.X\,(\lambda x.\text{saw}\,x\,y)}_{((\text{e}\to\text{t})\to\text{t})\to\text{e}\to\text{t}}$$

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

Any problems with this solution?

Lexicalism

$$[\![\text{saw}]\!] = \underbrace{\lambda X. \lambda y. X\,(\lambda x. \text{saw}\, x\, y)}_{((\text{e}\to\text{t})\to\text{t})\to\text{e}\to\text{t}}$$

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

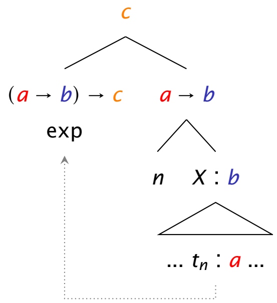
All the scopes!!

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:

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

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 *assignments*?
- Why didn't we pursue an applicative approach here? Could we?

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., $[\![3]\!] = \lambda k.\, k\, 3$, $[\![5]\!] = \lambda k.\, k\, 5$, $[\![+]\!] = \lambda k.\, k\,(+)$. Here's what it looks like to pass something its continuation:

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

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., $[\![3]\!] = \lambda k.\, k\, 3$, $[\![5]\!] = \lambda k.\, k\, 5$, $[\![+]\!] = \lambda k.\, k\,(+)$. Here's what it looks like to pass something its continuation:

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

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., $[\![3]\!] = \lambda k.\, k\, 3$, $[\![5]\!] = \lambda k.\, k\, 5$, $[\![+]\!] = \lambda k.\, k\,(+)$. Here's what it looks like to pass something its continuation:

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

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., $[\![3]\!] = \lambda k.\, k\, 3$, $[\![5]\!] = \lambda k.\, k\, 5$, $[\![+]\!] = \lambda k.\, k\,(+)$. Here's what it looks like to pass something its continuation:

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

The only difference here from a normal derivation built on simple functional application is that $[\![3]\!]$ is "the boss". We have *inverted control*.

Artist’s impression

The continuations applicative

As before, we start by characterizing the relevant notion of fanciness:[2]

$$\mathsf{C}\,a ::= (a \to \mathtt{t}) \to \mathtt{t}$$

And also as before, we then look for ρ and $\circledast$ with the right types:

$$\rho\, x := $$

[2] In fact, there is no real reason to artificially restrict ourselves to types of this form, but this will work well enough for the natural language examples we're interested in today.

The continuations applicative

As before, we start by characterizing the relevant notion of fanciness:[2]

$$\mathsf{C}\,a ::= (a \to \mathtt{t}) \to \mathtt{t}$$

And also as before, we then look for ρ and $\circledast$ with the right types:

$$\rho\, x := \lambda k.\, k\, x \qquad\qquad m \circledast n :=$$

[2] In fact, there is no real reason to artificially restrict ourselves to types of this form, but this will work well enough for the natural language examples we're interested in today.

The continuations applicative

As before, we start by characterizing the relevant notion of fanciness:[2]

$$\mathtt{C}\, a ::= (a \to \mathtt{t}) \to \mathtt{t}$$

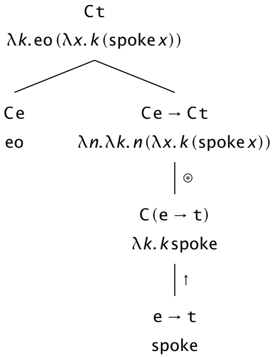
And also as before, we then look for ρ and $\circledast$ with the right types:

$$\rho\, x := \lambda k.\, k\, x \qquad\qquad m \circledast n := \lambda k.\, m(\lambda f.\, n(\lambda x.\, k\,(f\, x)))$$

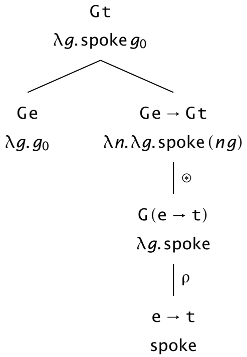
ρ is **LIFT** (‘$\uparrow$’)! $\circledast$ takes a continuized (scope-y) function and a continuized argument, scopes them, and delivers a continuized result.

[2] In fact, there is no real reason to artificially restrict ourselves to types of this form, but this will work well enough for the natural language examples we’re interested in today.

Sample derivation: *everyone spoke* (cf. *she$_0$ spoke*)



$\mathtt{C\,t}$
$\lambda k.\,\mathrm{eo}\,(\lambda x.\,k\,(\mathrm{spoke}\,x))$
$\mathtt{C\,e}$
eo
$\mathtt{C\,e \to C\,t}$
$\lambda n.\lambda k.\,n\,(\lambda x.\,k\,(\mathrm{spoke}\,x))$
$\circledast$
$\mathtt{C\,(e \to t)}$
$\lambda k.\,k\,\mathrm{spoke}$
$\uparrow$
$\mathtt{e \to t}$
spoke



Everyone spoke, in gory detail

$$\text{spoke}^{\uparrow} \circledast \text{eo} =$$

Everyone spoke, in gory detail

$$\begin{aligned} \mathsf{spoke}^{\uparrow} \circledast \mathsf{eo} &= \lambda k.\, \mathsf{spoke}^{\uparrow}(\lambda f.\, \mathsf{eo}(\lambda x.\, k\,(f\,x))) \\ &= \end{aligned}$$

Everyone spoke, in gory detail

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

Everyone spoke, in gory detail

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

Everyone spoke, in gory detail

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

Everyone spoke, in gory detail

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

Everyone spoke, in gory detail

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

Everyone spoke, in gory detail

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

The verb made its way under k, but part of the subject stayed above k. Which part?

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

The $\circledast$ 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*:

$$(\lambda k.\, A[k f]) \circledast (\lambda k.\, B[k x]) = \lambda k.\, A[B[k (f x)]]$$

This makes it easy to construct and reason about continuized derivations, despite the large 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 (quantifiers in object positions!):

$$\text{saw}^{\uparrow} \circledast \text{eo} =$$

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 (quantifiers in object positions!):

$$\text{saw}^{\uparrow} \circledast \text{eo} = \lambda k.\forall y.\, k\,(\text{saw}\, y)$$

Then, folding in the subject delivers... the inverse scope derivation! Why?

$$(\text{saw}^{\uparrow} \circledast \text{eo}) \circledast \text{so} = \lambda k.\forall y.\exists x.\, k\,(\text{saw}\, y\, x)$$

Surface scope?

We only derive inverse scope because $\circledast$ gives the "function-y" thing scope over the "argument-y" thing.

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

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

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

Can you think of arguments for or against this approach?

Against $\circledast'$?

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

1. Two people sent a letter to every student.

Which reading do you suppose that is?

Against $\circledast'$?

Sentences like the following have a reading that $\circledast$ and $\circledast'$ 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 $\circledast$ and $\circledast'$ can't generate it. Let's focus on how the VP and subject compose:

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

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

2 scope 2 far

1. (What was the party like?)

   Oh it was awful. Nobody came, and I had to clean up!

A sad vignette. What meaning does $\circledast$ derive for *nobody came*?

2 scope 2 far

1. (What was the party like?)

   Oh it was awful. Nobody came, and I had to clean up!

A sad vignette. What meaning does $\circledast$ 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:

2 scope 2 far

1. (What was the party like?)

   Oh it was awful. Nobody came, and I had to clean up!

A sad vignette. What meaning does $\circledast$ derive for *nobody came*?

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

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

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

Is this... ok?

2 scope 2 far

1. (What was the party like?)

   Oh it was awful. Nobody came, and I had to clean up!

A sad vignette. What meaning does $\circledast$ 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? 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” $\lambda k.\, \neg \exists x.\, k\, (\textbf{came}\, x)$?

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" $\lambda k.\, \neg\exists x.\, k\,(\textbf{came}\, x)$? Sure, turn it into something of type $\texttt{t}$ — i.e., find a "lowering" function, type $\mathsf{C}\,\texttt{t} \to \texttt{t}$.

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" $\lambda k.\, \neg\exists x.\, k\,(\textbf{came}\,x)$? Sure, turn it into something of type $\texttt{t}$ — i.e., find a "lowering" function, type $\texttt{C}\,\texttt{t} \to \texttt{t}$.

$$m^{\downarrow} = m\,(\lambda p.\, p)$$

An example of what this delivers for the present example:

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

Reset and scope islands

If we re-$\uparrow$ this, we complete something computer scientists call a **reset**[3]:

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

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

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

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

[3] E.g., Barker (2002), Charlow (2014).

Towers

The tower notation

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

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

A few examples of how this works:

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

The tower notation

Continuized derivations are easier to appreciate if we help outselves to an ingenious bit of notation known as **towers** (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 **towers** (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:

$$(\lambda k.\, A[k\, f]) \circledast (\lambda k.\, B[k\, x]) = \lambda k.\, A[B[k\,(f\, x)]]$$

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

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

Generally, I will omit $\circledast$ from now on: when you see two towers side by side, assume that $\circledast$ is gluing them together.

A basic example: *Mary saw everybody*

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

A basic example: *Mary saw everybody*

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

The derivation is immediate, and yields a result equivalent to what we'd obtain with a tedious series of β-reductions:

$$\lambda k.\forall x.\,k\,(\textbf{saw}\,x\,\textbf{m})$$

Two quantifiers: *somebody saw everybody*

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

Two quantifiers: *somebody saw everybody*

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

Again, the derivation is immediate, with the result equivalent to what we'd derive using a linear notation. From both perspectives, we see that scopal "effects" in functions are given priority over those in arguments.

$$\lambda k.\forall y.\exists x.\, k\,(\textbf{saw}\, y\, x)$$

Evaluation order

Shan & Barker (2006) make an interesting, and important suggestion. Instead of always giving *functions* priority. Why not always give *the thing on the left* priority?

$$\underbrace{\frac{A[\ ]}{x}}_{\text{l-exp}}\ \underbrace{\frac{B[\ ]}{y}}_{\text{r-exp}} \rightsquigarrow \frac{A[B[\ ]]}{x\,y \text{ or } y\,x\text{, whichever's defined}}$$

In other words, we do forwards or backwards functional application on the “bottom” story, whichever one works. The “effects” on the top levels are, however, consistently threaded together in their *linear* order.

Shan & Barker refer to this as “left-to-right evaluation”. It corresponds to a hypothesis that semantic evaluation has a built-in linear bias.

Somebody saw everybody

Now we can re-present the derivation of *somebody saw everybody*:

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

Somebody saw everybody

Now we can re-present the derivation of *somebody saw everybody*:

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

This time, we derive *surface scope*. We are no longer giving exclusive priority to the function, but to things on the left.

This seems like a better situation than the one we were in before — we bias surface-scope interpretations, just like speakers/hearers do. And the *explanation* for this bias is intuitive: people evaluate expressions in the order they are heard.

Varieties of lift

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

$$\frac{[\ ]}{\uparrow}\;\frac{\forall y.[\ ]}{y} \rightsquigarrow \begin{array}{c}\forall y.[\ ]\\ \hline [\ ]\\ \hline y\end{array}$$

As well as to towers:

$$\Bigg\uparrow \frac{\forall y.[\ ]}{y} \rightsquigarrow \begin{array}{c}[\ ]\\ \hline \forall y.[\ ]\\ \hline y\end{array}$$

Big tower combination

$$\begin{array}{c}A[\ ]\\\hline C[\ ]\\\hline f\end{array}\ \begin{array}{c}B[\ ]\\\hline D[\ ]\\\hline x\end{array}\ \rightsquigarrow\ \begin{array}{c}A[B[\ ]]\\\hline C[D[\ ]]\\\hline f\,x\end{array}$$

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

$$\left(\begin{array}{c}[\ ]\\\hline \\ \circledast\end{array}\ \begin{array}{c}A[\ ]\\\hline C[\ ]\\\hline f\end{array}\right)\begin{array}{c}B[\ ]\\\hline D[\ ]\\\hline x\end{array}\ =\ \begin{array}{c}A[B[\ ]]\\\hline C[D[\ ]]\\\hline f\,x\end{array}$$

And the same goes for towers of arbitary height.

Inverse scope

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

$$\begin{array}{c}[\;]\\ \hline \exists x.[\;]\\ \hline x\end{array}\left(\begin{array}{c}[\;]\\ \hline [\;]\\ \hline \textbf{saw}\end{array}\;\begin{array}{c}\forall y.[\;]\\ \hline [\;]\\ \hline y\end{array}\right) \rightsquigarrow \begin{array}{c}\forall y.[\;]\\ \hline \exists x.[\;]\\ \hline \textbf{saw}\,y\,x\end{array}$$

Lowering

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

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

Lowering our inverse-scope derivation:

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

A note on lowering

Lowering requires us to conjure an identity function. Along with $\uparrow$ and $\circledast$, 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 $\circledast$!

Barker, Chris. 2002. Continuations and the nature of quantification. *Natural Language Semantics* 10(3). 211–242. https://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. https://doi.org/10.3765/sp.1.1.

Charlow, Simon. 2014. *On the semantics of exceptional scope*. New York University Ph.D. thesis. http://semanticsarchive.net/Archive/2JmMWRjY/.

Charlow, Simon. 2017. A modular theory of pronouns and binding. In *Proceedings of Logic and Engineering of Natural Language Semantics 14*.

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

Kiselyov, Oleg. 2015. Applicative abstract categorial grammars. In Makoto Kanazawa, Lawrence S. Moss & Valeria de Paiva (eds.), *NLCS'15. Third workshop on natural language and computer science*, vol. 32 (EPiC Series), 29–38.

McBride, Conor & Ross Paterson. 2008. Applicative programming with effects. *Journal of Functional Programming* 18(1). 1–13. https://doi.org/10.1017/S0956796807006326.

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. https://doi.org/10.1007/BF00635836.

Shan, Chung-chieh & Chris Barker. 2006. Explaining crossover and superiority as left-to-right evaluation. *Linguistics and Philosophy* 29(1). 91–134. https://doi.org/10.1007/s10988-005-6580-7.

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