Continuations and monads

Computational semantics seminar

April 19, 2017

Today

We'll review what continuations are, and how continuized modes of composition offer a general account of scope-taking in natural language.

Then we'll zoom out. After all, whatever happened to monads?

Some things we'll look at in varying levels of detail:

- How does what we've learned about continuations interact with what we've learned about monads?
- Continuations characterize a type of compositional enrichment, much as monads do. So do continuations form a monad?
- If the answer's yes, does that fact turn out to be a *useful* one?

Continuations, reviewed

What are continuations?

Grammars based on continuations treat composition as oriented (or, more precisely, orient*able*) around scope and scope-taking.

- Barker (2002), Shan & Barker (2006), Barker & Shan (2014), . . .

This means taking an enrichments-based perspective on *quantification*, *scope ambiguity*, and so on — much as we have done all seminar long.

- We provide a way to combine two scopal meanings into a third.
- We provide a way to make meanings trivially scopal, so that everything can join in the fun.

Our two pieces of functional glue

Scope-enriched functional application:

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

Along with a “lifting” operation for making anything scopal:

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

There is nothing more to say! These two simple functions, one of them quite familiar, work to derive every single scope ordering in every case.

An example from last time

$$\textbf{saw}^{\uparrow} \mathbin{/\!\!/} \textbf{eo} =$$

An example from last time

$$
\begin{aligned}
\mathbf{saw}^{\uparrow} \mathbin{/\!\!/} \mathbf{eo} &= \lambda k.\, \mathbf{saw}^{\uparrow}(\lambda f.\, \mathbf{eo}(\lambda x.\, k\,(f\,x))) \\
&=
\end{aligned}
$$

An example from last time

$$\begin{aligned}
\textbf{saw}^{\uparrow} \mathbin{/\!/} \textbf{eo} &= \lambda k.\, \textbf{saw}^{\uparrow}(\lambda f.\, \textbf{eo}(\lambda x.\, k\,(f\,x))) \\
&= \lambda k.\, \textbf{saw}^{\uparrow}(\lambda f.\, (\lambda k'.\, \forall y.\, k'\,y)\,(\lambda x.\, k\,(f\,x))) \\
&=
\end{aligned}$$

An example from last time

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

An example from last time

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

An example from last time

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

An example from last time

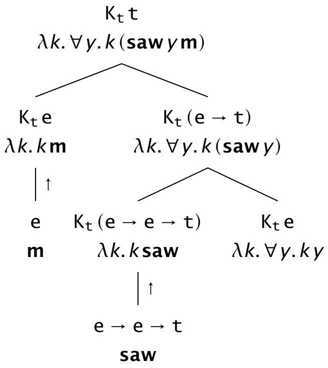
$$\begin{aligned}
\textbf{saw}^{\uparrow} \mathbin{/\!\!/} \textbf{eo} &= \lambda k.\, \textbf{saw}^{\uparrow}(\lambda f.\, \textbf{eo}(\lambda x.\, k\,(f\,x))) \\
&= \lambda k.\, \textbf{saw}^{\uparrow}(\lambda f.\, (\lambda k'.\, \forall y.\, k'\,y)\,(\lambda x.\, k\,(f\,x))) \\
&= \lambda k.\, \textbf{saw}^{\uparrow}(\lambda f.\, \forall y.\, (\lambda x.\, k\,(f\,x))\,y) \\
&= \lambda k.\, \textbf{saw}^{\uparrow}(\lambda f.\, \forall y.\, k\,(f\,y)) \\
&= \lambda k.\, (\lambda k'.\, k'\,\textbf{saw})(\lambda f.\, \forall y.\, k\,(f\,y)) \\
&= \lambda k.\, (\lambda f.\, \forall y.\, k\,(f\,y))\,\textbf{saw} \\
&=
\end{aligned}$$

An example from last time

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

The verb's meaning makes its way into the argument of k, while the irreducibly scopal part of the object ($\forall y$) stays outside k.

Mary saw everyone, in tree form


$\mathsf{K_t\,t}$
$\lambda k.\,\forall y.\,k\,(\mathbf{saw}\,y\,\mathbf{m})$
$\mathsf{K_t\,e}$
$\lambda k.\,k\,\mathbf{m}$
$\uparrow$
e
$\mathbf{m}$
$\mathsf{K_t\,(e \to t)}$
$\lambda k.\,\forall y.\,k\,(\mathbf{saw}\,y)$
$\mathsf{K_t\,(e \to e \to t)}$
$\lambda k.\,k\,\mathbf{saw}$
$\uparrow$
$\mathsf{e \to e \to t}$
$\mathbf{saw}$
$\mathsf{K_t\,e}$
$\lambda k.\,\forall y.\,k\,y$

Legend

$\mathsf{K}_r\,a ::= (a \to r) \to r$

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

A nice intuition

Here's something Anna Szabolcsi told me early in my graduate career, when I was trying to get a grip on this continuations business:

Continuations are just ubiquitous scopal pied-piping.

(Anna Szabolcsi, p.c.)

Why did Anna say this? Does it make sense to you?

Islands

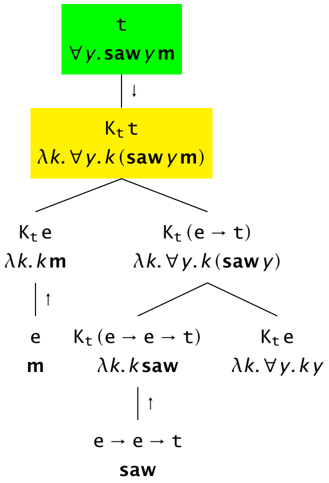
The **lowering** step is necessary to prevent rank over-generation.

Nobody came to the party, but I had to clean up cannot mean that nobody is such that: they came to the party, and I had to clean up.

Lowering closes off the scope of any quantifiers, preventing them from scoping beyond the level at which lowering applied:

$$\lambda k.\, \forall y.\, k\,(\mathbf{saw}\, y\, \mathbf{m}) \overset{\downarrow}{\Longrightarrow} \forall y.\, \mathbf{saw}\, y\, \mathbf{m} \overset{\uparrow}{\Longrightarrow} \lambda k.\, k\,(\forall y.\, \mathbf{saw}\, y\, \mathbf{m})$$

Mary saw everyone, lowered


t
$\forall y.\,\mathbf{saw}\,y\,\mathbf{m}$
$\downarrow$
$\mathrm{K_t\,t}$
$\lambda k.\forall y.\,k\,(\mathbf{saw}\,y\,\mathbf{m})$
$\mathrm{K_t\,e}$
$\lambda k.\,k\,\mathbf{m}$
$\mathrm{K_t\,(e \to t)}$
$\lambda k.\forall y.\,k\,(\mathbf{saw}\,y)$
$\uparrow$
e
$\mathbf{m}$
$\mathrm{K_t\,(e \to e \to t)}$
$\lambda k.\,k\,\mathbf{saw}$
$\mathrm{K_t\,e}$
$\lambda k.\forall y.\,k\,y$
$\uparrow$
$\mathrm{e \to e \to t}$
$\mathbf{saw}$

Legend
$\mathrm{K}_r\, a ::= (a \to r) \to r$
$x^{\uparrow} := \lambda k.\, k\, x$
$v^{\downarrow} := v\,(\lambda a.\, a)$

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 $/\!/$!

[This is a *small* fudge: letting identity functions run free in this way actually causes us to over-generate, without a slightly more sophisticated type theory.]

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 couple examples of how this works:

$$\lambda k.\, \forall y.\, k\,y \rightsquigarrow$$

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 couple examples of how this works:

$$\lambda k. \forall y. k\,y \rightsquigarrow \frac{\forall y. [\;]}{y} \qquad\qquad \frac{[\;]}{\textbf{saw}} \;/\!\!/\; \frac{\forall y. [\;]}{y} \rightsquigarrow$$

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 couple examples of how this works:

$$\lambda k.\, \forall y.\, k\,y \rightsquigarrow \frac{\forall y.\,[\;]}{y} \qquad\qquad \frac{[\;]}{\textbf{saw}} \;/\!\!/\; \frac{\forall y.\,[\;]}{y} \rightsquigarrow \frac{\forall y.\,[\;]}{\textbf{saw}\,y}$$

[I'll often suppress $/\!\!/$ in derivations to keep them simple.]

Deriving inverse scope

We can apply operations like ↑ to towers internally or externally:

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

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

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

Big tower combination

$$\cfrac{A[\ ]}{\cfrac{C[\ ]}{f}}\ \cfrac{B[\ ]}{\cfrac{D[\ ]}{x}} \rightsquigarrow \cfrac{A[B[\ ]]}{\cfrac{C[D[\ ]]}{f\,x}}$$

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

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

And the same goes for towers of arbitary height.

Big tower lowering

$$\dfrac{A[\;]}{\dfrac{B[\;]}{x}} \rightsquigarrow A[B[x]] \qquad\qquad \dfrac{\forall y.[\;]}{\dfrac{\exists x.[\;]}{\mathbf{saw}\,y\,x}} \rightsquigarrow \forall y.\exists x.\mathbf{saw}\,y\,x$$

Again, this rule already follows from the basic ones!

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

Big tower lowering

$$\frac{A[\;]}{\dfrac{B[\;]}{x}} \rightsquigarrow A[B[x]] \qquad \frac{\forall y.[\;]}{\dfrac{\exists x.[\;]}{\textbf{saw}\, y\, x}} \rightsquigarrow \forall y.\exists x.\textbf{saw}\, y\, x$$

Again, this rule already follows from the basic ones!

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

Big tower lowering

$$\begin{array}{c} A[\ ] \\ \hline B[\ ] \\ \hline x \end{array} \rightsquigarrow A[B[x]] \qquad\qquad \begin{array}{c} \forall y.[\ ] \\ \hline \exists x.[\ ] \\ \hline \mathbf{saw}\, y\, x \end{array} \rightsquigarrow \forall y.\exists x.\mathbf{saw}\, y\, x$$

Again, this rule already follows from the basic ones!

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

Monads and continuations

Mother!

> *[T]he continuation monad is in some sense **the mother of all monads**.*
>
> (Unknown, *apud* Dan Piponi, emphasis mine)

You probably have a couple questions at this point:

- What is the continuation **monad**? We haven't seen any such beast!
- What is this stuff about mother? Seems weird and obscure. . .

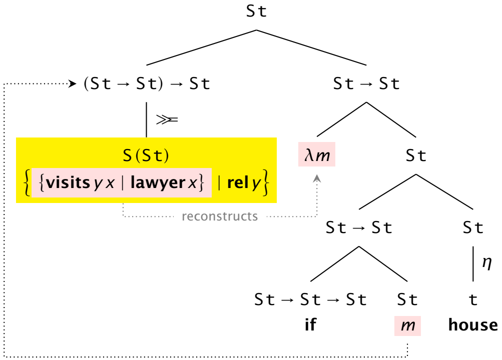
Some other questions you might have

We've only seen how to use continuations to implement scope-taking for rather boring, vanilla things like boolean generalized quantifiers.

But much of what we've done in this seminar involves scope-taking of far more interesting things: alternative-generating expressions, things with binding potential, dynamic meanings, supplemental content, etc.

How (if at all) do these things fit into the continuations picture?

What do(es) continuations have to say about this??



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

The type of $\gg=$

Recall the type we assign to a monadic $\gg=$ operation:

$$T\,a \to (a \to T\,b) \to T\,b$$

Alongside a diagram that seems somehow familiar...

$$\begin{tikzcd}
a \arrow[r, "\mathbf{LIFT}_{T\,b}"] \arrow[d, "\eta"'] & (a \to T\,b) \to T\,b \\
T\,a \arrow[ur, "\gg="] &
\end{tikzcd}$$

The type of $m^{\gg=}$

Given an $m :: T\,a$, $m^{\gg=}$ will have the following type:

$$(a \to T\,b) \to T\,b$$

I hate to ask, but... does this remind you of anything?

The type of $m^{\ggcurly}$

Given an $m :: T\,a$, $m^{\ggcurly}$ will have the following type:

$$(a \to T\,b) \to T\,b$$

I hate to ask, but... does this remind you of anything? It looks a helluva lot like the type of a generalized quantifier!!!

And indeed, we observe that the continuations type constructor is directly applicable to such cases:

$$(a \to T\,b) \to T\,b \equiv \mathrm{K}_{T\,b}\,a$$

So instead of restricting ourselves to “pure” **return types**, we might as well broaden our views to monadic return types.

In other words

Any $m^{\ggcurly}$ is *on the hunt for a scope argument* — i.e., a continuation:

$$m^{\ggcurly} = \lambda k.\, m \ggcurly k$$

Can you see how to convert this into a tower?

$$\lambda k.\, m \ggcurly k \;=$$

In other words

Any $m^{\gg=}$ is *on the hunt for a scope argument* — i.e., a continuation:

$$m^{\gg=} = \lambda k.\, m \gg= k$$

Can you see how to convert this into a tower?

$$\lambda k.\, m \gg= k \;=\; \lambda k.\, m \gg= \lambda x.\, k x \;\rightsquigarrow$$

In other words

Any $m^{\ggg}$ is *on the hunt for a scope argument* — i.e., a continuation:

$$m^{\ggg} = \lambda k.\, m \ggg k$$

Can you see how to convert this into a tower?

$$\lambda k.\, m \ggg k \;=\; \lambda k.\, m \ggg \lambda x.\, k\, x \;\rightsquigarrow\; \frac{m \ggg \lambda x.\, [\;]}{x}$$

A concrete case: alternatives

$$\frac{[\ ]}{\mathbf{j}}\left(\frac{[\ ]}{\mathbf{saw}}\ \frac{\{x \mid \mathbf{ling}\, x\} \ggcurly= \lambda y.[\ ]}{y}\right) \rightsquigarrow \frac{\{x \mid \mathbf{ling}\, x\} \ggcurly= \lambda y.[\ ]}{\mathbf{saw}\, y\, \mathbf{j}}$$

Tower combination simply works on *anything* that takes scope. It is a perfectly general solution to the problem of scope-taking.

Nevertheless, a couple of questions arise at this point:

A concrete case: alternatives

$$\frac{[\ ]}{\mathbf{j}} \left(\frac{[\ ]}{\mathbf{saw}} \; \frac{\{x \mid \mathbf{ling}\, x\} \gg\!= \lambda y.[\ ]}{y} \right) \rightsquigarrow \frac{\{x \mid \mathbf{ling}\, x\} \gg\!= \lambda y.[\ ]}{\mathbf{saw}\, y\, \mathbf{j}}$$

Tower combination simply works on *anything* that takes scope. It is a perfectly general solution to the problem of scope-taking.

Nevertheless, a couple of questions arise at this point:

- Whence the trivial towers? The answer we're used to at this point is $\uparrow$, i.e., **LIFT**. Does another possibility present itself?
- To end a derivation, i.e., enforce scope islands, we need to collapse the tower via lower. Does our old trick work for that?

On $\uparrow$/**LIFT**

Think back to the first monad law (or, if you like, the manifold triangle):

$$\eta\, x \ggg k = k\, x$$

What does this tell you about the nature of **LIFT**?

On $\uparrow$/**LIFT**

Think back to the first monad law (or, if you like, the manifold triangle):

$$\eta\, x \gg\!= k = k\, x$$

What does this tell you about the nature of **LIFT**? **LIFT** is a *derivative* operation in the presence of monads!

$$(\eta\, x)^{\gg=} \;=$$

On ↑/**LIFT**

Think back to the first monad law (or, if you like, the manifold triangle):

$$\eta\, x \gg= k = k\, x$$

What does this tell you about the nature of **LIFT**? **LIFT** is a *derivative* operation in the presence of monads!

$$(\eta\, x)^{\gg=} \;=\; \lambda k.\, \eta\, x \gg= k \;=$$

On ↑/**LIFT**

Think back to the first monad law (or, if you like, the manifold triangle):

$$\eta\, x \gg\!= k = k\, x$$

What does this tell you about the nature of **LIFT**? **LIFT** is a *derivative* operation in the presence of monads!

$$(\eta\, x)^{\gg=} \;=\; \lambda k.\, \eta\, x \gg\!= k \;=\; \lambda k.\, k\, x \;=$$

On ↑/**LIFT**

Think back to the first monad law (or, if you like, the manifold triangle):

$$\eta\, x \ggg\!= k = k\, x$$

What does this tell you about the nature of **LIFT**? **LIFT** is a *derivative* operation in the presence of monads!

$$(\eta\, x)^{\ggg=} \;=\; \lambda k.\, \eta\, x \ggg\!= k \;=\; \lambda k.\, k\, x \;=\; \frac{[\ ]}{x}$$

On $\downarrow$

In the standard continuations framework, $\downarrow$ involves application to an identity function. Will that work for the monadic case?

$$\frac{\{x \mid \mathbf{ling}\, x\} \ggg \lambda y.[\;]}{\mathbf{saw}\, y\, \mathbf{j}} = \lambda k.\{x \mid \mathbf{ling}\, x\} \ggg \lambda y.k\,(\mathbf{saw}\, y\, \mathbf{j})$$

No! If we attempt to apply this tower to $\lambda a.a$, the result isn't well-typed!

$$\{x \mid \mathbf{ling}\, x\} \ggg \lambda y.\mathbf{saw}\, y\, \mathbf{j}$$

What should we try instead? What other trivial continuation do we have access to, which will result in something well-typed?

$\downarrow$ as application to η

$$\begin{aligned}(\lambda k.\,\{x \mid \mathbf{ling}\,x\} \ggcurly \lambda y.\,k\,(\mathbf{saw}\,y\,\mathbf{j}))\,\eta &= \{x \mid \mathbf{ling}\,x\} \ggcurly \lambda y.\,\eta\,(\mathbf{saw}\,y\,\mathbf{j}) \\ &= \{\mathbf{saw}\,y\,\mathbf{j} \mid \mathbf{ling}\,y\}\end{aligned}$$

Thus, lowering in a monadic setting simply corresponds to application to a different sort of trivial continuation or scope: η.

Summing up

The monadic operations of η and $\gg\!=$ are *decompositions* of the operations that undergird continuized composition:

$$\uparrow \; = \; \gg\!= \circ \, \eta \qquad\qquad m^{\downarrow} = m\,\eta$$

But nothing else about the continuized grammar needs to change. In particular, $/\!\!/$ is still the way that scopal meanings are built up.

More generally, this works for any monad *whatsoever*:

$$\frac{(\lambda g.\, g_0) \gg\!= \lambda x.\,[\;]}{x} \qquad \frac{(\mathbf{j}, \mathbf{ling}\,\mathbf{j}) \gg\!= \lambda x.\,[\;]}{x} \qquad \frac{(\lambda g.\, \{(y, g^{0\to y}) \mid \mathbf{ling}\, y\}) \gg\!= \lambda x.\,[\;]}{x}$$

Classical continuations

Given that continuized composition can be driven by an arbitrary "underlying" monad, is there a monad that classical continuized composition corresponds to?

Consider that for classical continuations, lowering involves application to the identity function. Is there a monad for which $\eta = \lambda a. a$? Indeed there is, and it's called the **Identity** monad:

$$\eta\, x = x \qquad\qquad m \ggg k = k\, m$$

Double-checking that $\ggg \circ\, \eta =$ **LIFT**:

$$(\eta\, x)^{\ggg} = ((\lambda a. a)\, x)^{\ggg} = x^{\ggg} = \lambda k. k\, x$$

Zooming out

This cashes out a promissory note that we've been carrying with us all seminar long. In particular, I've frequently appealed to things that look like LFs, while telling you to not necessarily think of them as literal LFs.

One issue, recall, what that in LFs, traces are interpreted as variables, which seems to throw a wrench in the "pure-impure" separation that drives monadic composition (the trace would need to take scope!).

We see now that there was no cause for concern. I wasn't pulling a fast one on you. If scope-taking is via continuations, there is no awkwardness at all in supposing that monadic composition is driven by scope.

Zooming further out

Islands have been a key touchstone for the past 14-odd weeks.

In a continuized theory, an island is a *semantic phase*, a domain that have to be obligatorily lowered (i.e., **evaluated**). Consider in that respect what happens when we re-“lift” a monadic island:

$$\{\textbf{saw}\,y\,\textbf{j} \mid \textbf{ling}\,y\}^{\gg=} = \lambda k.\,\{\textbf{saw}\,y\,\textbf{j} \mid \textbf{ling}\,y\} \gg= k = \frac{\{\textbf{saw}\,y\,\textbf{j} \mid \textbf{ling}\,y\} \gg= \lambda p.[\ ]}{p}$$

The result here is equivalent to the following:

$$\frac{\{x \mid \textbf{ling}\,x\} \gg= \lambda y.[\ ]}{\textbf{saw}\,y\,\textbf{j}}$$

Mother?

So in what sense are continuations the *mother* of all monads?

What this means is that continuations provide a highly general interface for *doing composition with monads*.

Because continuations are just a denotational reification of scope-taking, we might as well say that QR is “the mother of all monads”!

Other modes of composition

Before we move on, I want to point out a neat further generalization of the basic approach here. Recall our $\sslash$ operation:

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

As we have just seen, this works even when F and X are monadic. Can you think of a further way to generalize it?

Instead of doing functional application at the end of the day, we might do something else! How about... predicate modification?

$$F \curlywedge G := \lambda k.\, F\,(\lambda f.\, G\,(\lambda g.\, k\,(\lambda e.\, f\,e \wedge g\,e)))$$

Scopal modification, with towers:

$$\frac{A[\ ]}{f}\ \frac{B[\ ]}{g} \rightsquigarrow \frac{A[B[\ ]]}{\lambda e.f\,e \wedge g\,e}$$

What’s interesting about this?

Because scopal composition can be generalized to *any* underlying compositional operation (modification, application, etc etc), we have an immediate integration of scope-taking with **event semantics**.

It has been suggested that such an integration can be non-trivial to achieve (Champollion 2015). In the presence of continuations, it looks very nearly trivial!

Moreoever, because continuations generalize to any monad, we also have monadic composition with event semantics, if we want it.

Example derivation

$$\frac{[\;]}{\lambda f.\exists e.f\,e}\left(\left(\left(\frac{[\;]}{\lambda x.\lambda e.\mathbf{Ag}_e = x}\;\frac{[\;]}{\mathbf{j}}\right)\left(\frac{[\;]}{\mathbf{saw}}\;\left(\frac{[\;]}{\lambda y.\lambda e.\mathbf{Th}_e = y}\;\frac{\forall y.[\;]}{y}\right)\right)\right)\right)$$

$$\rightsquigarrow \forall y.\exists e.\mathbf{Ag}_e = \mathbf{j} \wedge \mathbf{saw}\,e \wedge \mathbf{Th}_e = y$$

Thinking (more) categorically

Various constructs (redux)

We have seen several constructs for characterizing how “enriched” meanings interact with others in semantic composition.

- **Monads** allow us to sequence an enriched thing with a scope:

$$\gg\!= :: T\,a \to (a \to T\,b) \to T\,b$$

- **Applicatives** give rise to a notion of enriched functional application:

$$\blacktriangleright :: T\,a \to T\,(a \to b) \to T\,b$$

- **Functors** allow us to map a function over some structure:

$$\triangleright :: T\,a \to (a \to b) \to T\,b$$

Relative strength (redux)

Every monad is applicative, and every applicative is a functor:

$$\textbf{Monad} \subset \textbf{Applicative} \subset \textbf{Functor}$$

To wit, given a monadic $\ggcurly$, $\blacktriangleright$ and $\triangleright$ can be recovered as follows:

$$m \blacktriangleright n := m \ggcurly \lambda x.\, n \ggcurly \lambda f.\, \eta\,(f\,x)$$

$$m \triangleright f := m \ggcurly \lambda x.\, \eta\,(f\,x)$$

What do we have?

Recall our type constructor for continuized meanings:

$$\mathrm{K}_r\, a ::= (a \to r) \to r$$

Our two core operations are given as follows:

$$\uparrow\ ::\ a \to \mathrm{K}_r\, a \qquad\qquad /\!\!/\ ::\ \mathrm{K}_r\, a \to \mathrm{K}_r\,(a \to b) \to \mathrm{K}_r\, b$$

So we are in the presence of a(n). . .

What do we have?

Recall our type constructor for continuized meanings:

$$\mathsf{K}_r\, a ::= (a \to r) \to r$$

Our two core operations are given as follows:

$$\uparrow\ ::\ a \to \mathsf{K}_r\, a \qquad\qquad /\!\!/\ ::\ \mathsf{K}_r\, a \to \mathsf{K}_r\,(a \to b) \to \mathsf{K}_r\, b$$

So we are in the presence of a(n)... **applicative**.

Applicative definition

An applicative is a type constructor T with two functions:

$$\eta :: a \to T\,a \qquad \circledast :: T\,(a \to b) \to T\,a \to T\,b$$

Satisfying a few laws:

Homomorphism

$\eta\,f \circledast \eta\,x =$

Identity

Applicative definition

An applicative is a type constructor T with two functions:

$$\eta :: a \to T\,a \qquad \circledast :: T\,(a \to b) \to T\,a \to T\,b$$

Satisfying a few laws:

Homomorphism

$\eta\,f \circledast \eta\,x = \eta\,(f\,x)$

Identity

$\eta\,(\lambda x.x) \circledast v =$

Applicative definition

An applicative is a type constructor T with two functions:

$$\eta :: a \to T\,a \qquad \circledast :: T\,(a \to b) \to T\,a \to T\,b$$

Satisfying a few laws:

Homomorphism

$\eta\,f \circledast \eta\,x = \eta\,(f\,x)$

Identity

$\eta\,(\lambda x.\,x) \circledast v = v$

Interchange

$\eta\,(\lambda f.\,f\,x) \circledast u =$

Composition

Applicative definition

An applicative is a type constructor T with two functions:

$$\eta :: a \to T\,a \qquad \circledast :: T\,(a \to b) \to T\,a \to T\,b$$

Satisfying a few laws:

Homomorphism

$\eta\,f \circledast \eta\,x = \eta\,(f\,x)$

Identity

$\eta\,(\lambda x.x) \circledast v = v$

Interchange

$\eta\,(\lambda f.f\,x) \circledast u = u \circledast \eta\,x$

Composition

$\eta\,(\circ) \circledast u \circledast v \circledast w =$

Applicative definition

An applicative is a type constructor T with two functions:

$$\eta :: a \to T\,a \qquad \mathbin{/\!/} :: T\,(a \to b) \to T\,a \to T\,b$$

Satisfying a few laws:

Homomorphism

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

Identity

$\eta\,(\lambda x.\, x) \mathbin{/\!/} v = v$

Interchange

$\eta\,(\lambda f.\, f\,x) \mathbin{/\!/} u = u \mathbin{/\!/} \eta\, x$

Composition

$\eta\,(\circ) \mathbin{/\!/} u \mathbin{/\!/} v \mathbin{/\!/} w = u \mathbin{/\!/} (v \mathbin{/\!/} w)$

Basically, these laws say that $\mathbin{/\!/}$ should be a kind of fancy functional application, and η should be a trivial way to make something fancy.

Other applicatives

We have seen many instances of applicatives in the seminar (because we have seen many monads!). We have also seen one or two cases of applicatives that aren't monads. The following example springs to mind:

$$F \mathbin{/\!\!/} X := \{\lambda g.\, f\, g\, (x\, g) \mid f \in F, x \in X\}$$

[cf. Romero & Novel 2013]

Exercise: convince yourself that this applicative (and the other ones we've seen) satisfies the applicative laws!

Applicatives compose!

If you have two monadic type constructors S and T, you aren't guaranteed to have a composite monad $S \circ T$ or $T \circ S$.

But whenever you have two applicative constructors S and T, both $S \circ T$ and $T \circ S$ are guaranteed to be applicative (McBride & Paterson 2008)!

We just saw a concrete example:

$$\mathrm{G}\,a ::= \mathrm{g} \rightarrow a \qquad \mathrm{S}\,a ::= a \rightarrow \mathrm{t}$$

While assignment-dependent alternative sets (G ∘ S) are a monad, sets of assignment-dependent meanings (S ∘ G) aren't!

Do we also have a monad?

What’s the type constructor?

Do we also have a monad?

What's the type constructor?

$$\mathsf{K}_r\, a ::= (a \to r) \to r$$

And now how about the η and $\ggcurly$ operations?

Do we also have a monad?

What's the type constructor?

$$\mathsf{K}_r\, a ::= (a \to r) \to r$$

And now how about the η and $\ggcurly$ operations?

$$\eta\, x = x^{\uparrow}$$

Do we also have a monad?

What's the type constructor?

$$\mathsf{K}_r\, a ::= (a \to r) \to r$$

And now how about the η and $\gg\!=$ operations?

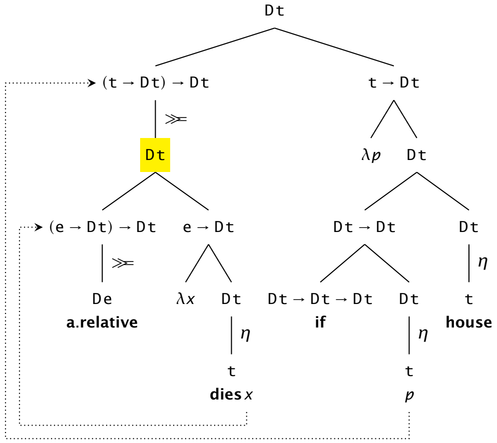
$$\eta\, x = x^{\uparrow} \qquad\qquad m \gg\!= f = \lambda k.\, m\,(\lambda x.\, f\, x\, k)$$

Check for yourself that these satisfy the three monad laws!

[Incredibly, this $\gg\!=$ is identical in form, though not in type, to the +WH shifter from Cresti (1995), cf. also Karttunen (1977)!!]

Is the monad *useful?*

Remember that monads require **scope** to do their magic.



An ouroboros

None of that tree could be composed using η and $\gg\!=$ alone!

We **need** scope-taking in addition!

So the continuation monad would require some extra apparatus for. . .
scope!!! Which would make it explanatorily idle.

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.

Barker, Chris & Chung-chieh Shan. 2014. *Continuations and natural language.* Oxford: Oxford University Press. http://dx.doi.org/10.1093/acprof:oso/9780199575015.001.0001.

Champollion, Lucas. 2015. The interaction of compositional semantics and event semantics. *Linguistics and Philosophy* 38(1). 31–66. http://dx.doi.org/10.1007/s10988-014-9162-8.

Cresti, Diana. 1995. Extraction and reconstruction. *Natural Language Semantics* 3(1). 79–122. http://dx.doi.org/10.1007/BF01252885.

Karttunen, Lauri. 1977. Syntax and semantics of questions. *Linguistics and Philosophy* 1(1). 3–44. http://dx.doi.org/10.1007/BF00351935.

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

Romero, Maribel & Marc Novel. 2013. Variable binding and sets of alternatives. In Anamaria Fălăuș (ed.), *Alternatives in Semantics*, chap. 7, 174–208. London: Palgrave Macmillan UK. http://dx.doi.org/10.1057/9781137317247_7.

References II

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