Dynamic alternatives

Computational semantics seminar

February 15, 2017

Dynamic semantics

Basic data

Indefinites behave more like proper names than quantifiers with respect to (e.g.) discourse anaphora:

1. {Polly, a linguist, *every linguist, *no linguist}^{n} entered. She$_{n}$ sat.

Problems of static semantics

Such data is difficult to explain if we treat indefinites as quantifiers!

Quantifiers need to scope over any pronouns they bind. One might like to just bite the bullet and say that cross-sentential binding by indefinites is some kind of text-level scope, but that makes bad predictions:

1. Exactly one linguist^n entered. She$_n$ sat.
2. Exactly one linguist^n entered and sat.

Sentences (1) and (2) don't mean the same things. So discourse anaphora cannot be a simple matter of text-level scope-taking.

PLA by induction

Let's get a feel for how a basic dynamic system works by playing with one. Here's a toy implementation of Dekker's (1994) PLA. (Put together by Dylan Bumford and me.)

What do you notice?

PLA by induction

Let's get a feel for how a basic dynamic system works by playing with one. Here's a toy implementation of Dekker's (1994) PLA. (Put together by Dylan Bumford and me.)

What do you notice? PLA treats a context as a set of possibilities tracking information about the values of pronouns.

Basic PLA pieces

Meanings for simple “sentences” (adopting a post-fix notation):

$$G[\mathbf{left}\, v] :=$$

Basic PLA pieces

Meanings for simple “sentences” (adopting a post-fix notation):

$$G[\mathbf{left}\, v] := \{g \in G \mid \mathrm{left}\, g_v\}$$

Meanings for dynamic existential quantifiers:

$$G[\exists^v] :=$$

Basic PLA pieces

Meanings for simple "sentences" (adopting a post-fix notation):

$$G[\mathbf{left}\, v] := \{g \in G \mid \mathrm{left}\, g_v\}$$

Meanings for dynamic existential quantifiers:

$$G[\exists^v] := \{g^{v \to x} \mid g \in G, x :: \mathrm{e}\}$$

Meaning for dynamic conjunction:

$$G[p\,;q] :=$$

Basic PLA pieces

Meanings for simple "sentences" (adopting a post-fix notation):

$$G[\mathbf{left}\, v] := \{g \in G \mid \mathrm{left}\, g_v\}$$

Meanings for dynamic existential quantifiers:

$$G[\exists^v] := \{g^{v \to x} \mid g \in G, x :: \mathrm{e}\}$$

Meaning for dynamic conjunction:

$$G[p\,;q] := G[p][q]$$

Getting closure

How can we extract a normal static truth-condition from a PLA meaning?

$$\lightning :: \mathsf{S}\,\mathsf{g} \to \mathsf{g} \to \mathsf{t}$$

$$p^{\lightning} :=$$

Getting closure

How can we extract a normal static truth-condition from a PLA meaning?

$$\lightning :: \mathtt{S}\,\mathtt{g} \to \mathtt{g} \to \mathtt{t}$$
$$p^{\lightning} := \lambda g.\, \exists h \in \{g\}\,[p]$$

Conjunction is associative

PLA conjunction is an **associative** opertion:

$$(p \,;\, q) \,;\, r = p \,;\, (q \,;\, r)$$

This isn't difficult to prove:

$$(p \,;\, q) \,;\, r =$$

Conjunction is associative

PLA conjunction is an **associative** opertion:

$$(p \,;\, q) \,;\, r = p \,;\, (q \,;\, r)$$

This isn't difficult to prove:

$$(p \,;\, q) \,;\, r = (\lambda s.\, q\,(p\,s)) \,;\, r$$

Conjunction is associative

PLA conjunction is an **associative** opertion:

$$(p\,;q)\,;r = p\,;(q\,;r)$$

This isn't difficult to prove:

$$(p\,;q)\,;r = (\lambda s.\,q\,(p\,s))\,;r \qquad p\,;(q\,;r) =$$
$$= \lambda s.\,r\,(q\,(p\,s))$$

Conjunction is associative

PLA conjunction is an **associative** opertion:

$$(p\,;q)\,;r = p\,;(q\,;r)$$

This isn't difficult to prove:

$$(p\,;q)\,;r = (\lambda s.\,q\,(p\,s))\,;r \qquad p\,;(q\,;r) = p\,;(\lambda s.\,r\,(q\,s))$$
$$= \lambda s.\,r\,(q\,(p\,s))$$

Conjunction is associative

PLA conjunction is an **associative** opertion:

$$(p\,;q)\,;r = p\,;(q\,;r)$$

This isn't difficult to prove:

$$\begin{aligned}(p\,;q)\,;r &= (\lambda s.\,q\,(p\,s))\,;r \\ &= \lambda s.\,r\,(q\,(p\,s))\end{aligned}$$

$$\begin{aligned}p\,;(q\,;r) &= p\,;(\lambda s.\,r\,(q\,s)) \\ &= \lambda s.\,r\,(q\,(p\,s))\end{aligned}$$

Notice how this fact is key to generating cross-sentential anaphora! Even in $(p\,;q)\,;r$, r is evaluated relative to the context established by p!

Going point-wise

Just decomposing the all at once update into pieces

$$\lambda i.\, \{ i^{n \to x} \mid \mathbf{ling}\, x, \mathbf{entered}\, x \}$$

And then instead of interpreting conjunction as function composition, interpret it as relation composition (still associative!).

$$L \,;\, R :=$$

Going point-wise

Just decomposing the all at once update into pieces

$$\lambda i.\, \{ i^{n \to x} \mid \mathbf{ling}\, x, \mathbf{entered}\, x \}$$

And then instead of interpreting conjunction as function composition, interpret it as relation composition (still associative!).

$$L \,;\, R := \lambda i.$$

Going point-wise

Just decomposing the all at once update into pieces

$$\lambda i.\, \{ i^{n \to x} \mid \mathbf{ling}\, x, \mathbf{entered}\, x \}$$

And then instead of interpreting conjunction as function composition, interpret it as relation composition (still associative!).

$$L \,;\, R := \lambda i. \bigcup_{j \in L\, i}$$

Going point-wise

Just decomposing the all at once update into pieces

$$\lambda i.\, \{ i^{n \to x} \mid \mathbf{ling}\, x, \mathbf{entered}\, x \}$$

And then instead of interpreting conjunction as function composition, interpret it as relation composition (still associative!).

$$L \,;\, R := \lambda i. \bigcup_{j \in L\, i} R j$$

Here, update vs. point-wise makes no difference. See Charlow (2016) for a case where the choice matters.

The standard account

In **dynamic semantics**, sentence meanings store values for variables:

$$[\![\text{Polly}^n \text{ entered}]\!] = \lambda g.\{g^{n \to \mathbf{p}} \mid \textbf{entered}\,\mathbf{p}\}$$

$$[\![\text{a linguist}^n \text{ entered}]\!] = \lambda g.\{g^{n \to x} \mid \textbf{entered}\,x\}$$

Type: $\mathsf{g} \to \mathsf{S}\,\mathsf{g}$

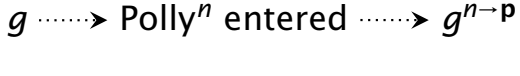
Compare the corresponding static meanings:

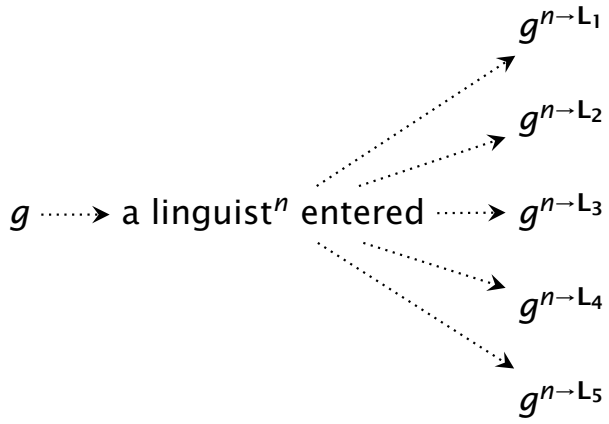
$$[\![\text{Polly}^n \text{ entered}]\!] = \lambda g.\textbf{entered}\,\mathbf{p}$$

$$[\![\text{a linguist}^n \text{ entered}]\!] = \lambda g.\textbf{entered}\,x$$

Type: $\mathsf{g} \to \mathsf{t}$

Pictorially



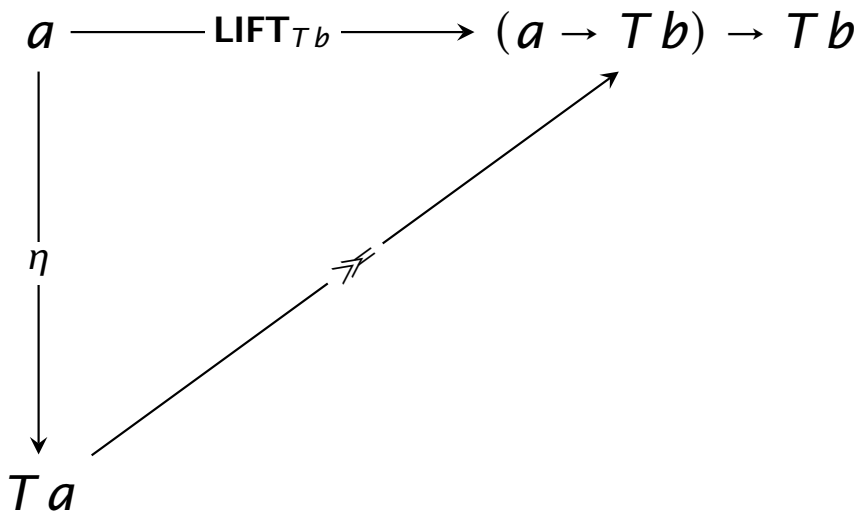


Monadic dynamic semantics

Again with the triangle?

A **monad** is a type constructor T associated with two functions, $\eta :: a \to T\,a$ and $\gg\!= :: T\,a \to (a \to T\,b) \to T\,b$, with three properties:

$$(\eta\,x) \gg\!= f = f\,x \qquad (\textit{Left Id})$$
$$m \gg\!= \eta = m \qquad (\textit{Right Id})$$
$$(m \gg\!= f) \gg\!= g = m \gg\!= \lambda x.\, f\,x \gg\!= g \qquad (\textit{Assoc})$$



Is standard dynamic semantics monadic? Points:

Both theories are oriented around things you might broadly think of as **exceptional scope** phenomena (quantificational vs. anaphoric).

Associativity is a key feature of how dynamic binding is secured. A kind of associativity also characterizes monadic $\gg=$ operations!

Alternatives play a central role in both alternative semantics (obvi) and dynamic semantics (where indefinites introduce alternative assignments)! And both theories treat indefinites as **non-quantificational**.

Is standard dynamic semantics monadic? Counterpoints:

Composition monadic settings is **enriched** composition, facilitated by η and $\gg=$. By contrast, composition in dynamic semantics is **straight functional application**.

Dynamic semantics only associates "dynamic effects" with **sentence-sized** chunks of structure. But monadic theories potentially associate "monadic effects" with anything down to the morpheme level.

Dynamic semantics has nothing to say about explain exceptional **quantificational** scope. And the version of monadic alternative semantics we've seen says nothing about exceptional **binding** scope.

More generally

The formal resources used by (monadic) alternatives-oriented theories and dynamic theories are so different as to seem irreconcilable!

Reminder: A tale of 2 monads

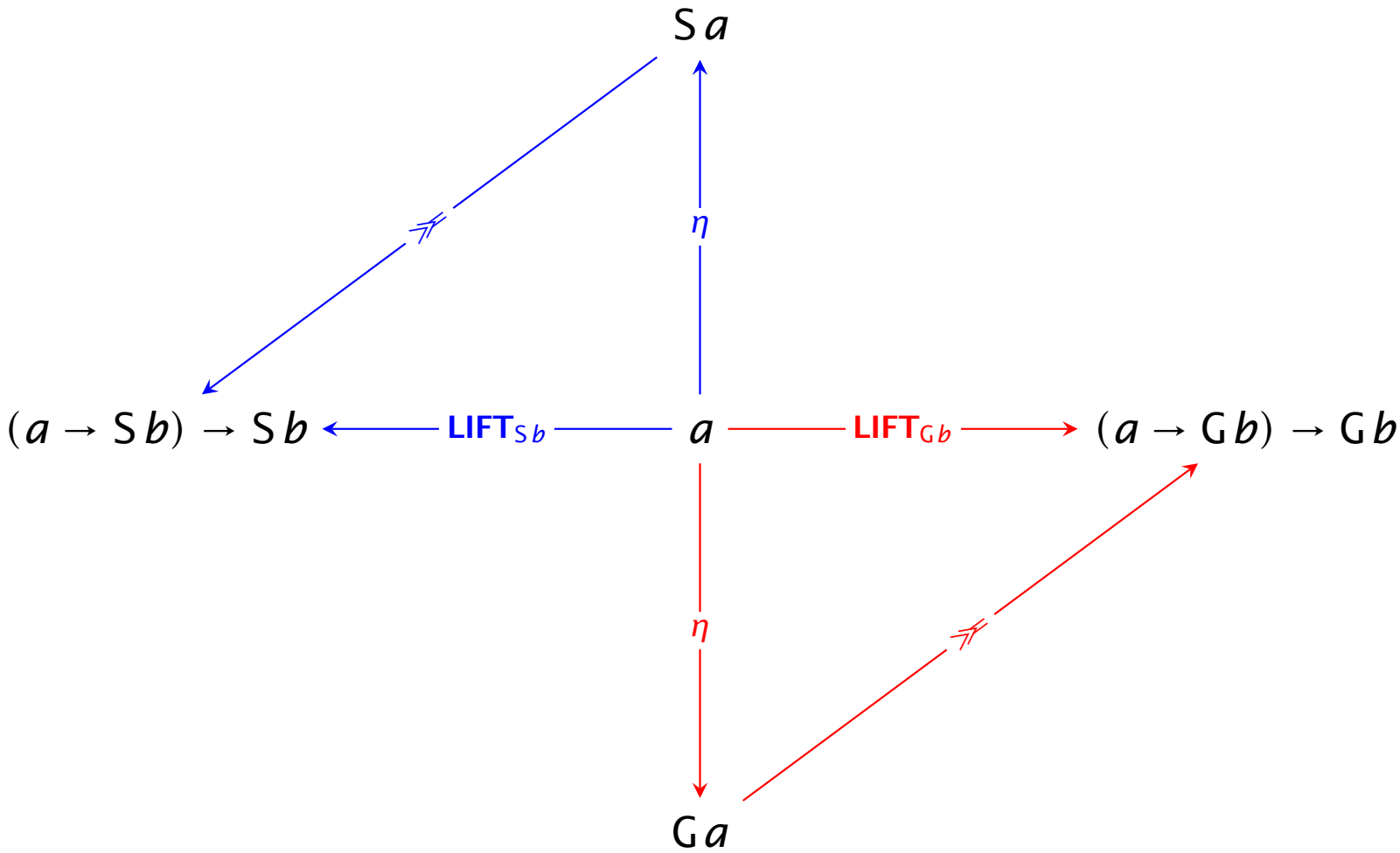
Two monads, one for alternatives....

$$\begin{array}{ll} \eta \;\; :: \; a \to \mathsf{S}\, a & \gg\!= \;\; :: \; \mathsf{S}\, a \to (a \to \mathsf{S}\, b) \to \mathsf{S}\, b \\ \eta\, x := \{x\} & m \gg\!= f := \bigcup_{x \in m} f\, x \end{array}$$

...And one for binding:

$$\begin{array}{ll} \eta \;\; :: \; a \to \mathsf{G}\, a & \gg\!= \;\; :: \; \mathsf{G}\, a \to (a \to \mathsf{G}\, b) \to \mathsf{G}\, b \\ \eta\, x := \lambda g.\, x & m \gg\!= f := \lambda g.\, f\, (m\, g)\, g \end{array}$$

How they [don't] relate...



Finding a composite monad

Two possibilities for layering assignments and alternatives:

$$\mathsf{GS}\,a ::= \mathsf{G}\,(\mathsf{S}\,a) \qquad \mathsf{SG}\,a ::= \mathsf{S}\,(\mathsf{G}\,a)$$

The first, but not the second, gives rise to a monad:

$$\eta \quad :: \; a \to \mathsf{GS}\,a$$

Finding a composite monad

Two possibilities for layering assignments and alternatives:

$$\mathsf{GS}\,a ::= \mathsf{G}\,(\mathsf{S}\,a) \qquad \mathsf{SG}\,a ::= \mathsf{S}\,(\mathsf{G}\,a)$$

The first, but not the second, gives rise to a monad:

$$\begin{aligned} \eta &:: a \to \mathsf{GS}\,a \\ \eta\,x &:= \lambda g.\,\{x\} \end{aligned}$$

Finding a composite monad

Two possibilities for layering assignments and alternatives:

$$\mathsf{GS}\,a ::= \mathsf{G}\,(\mathsf{S}\,a) \qquad\qquad \mathsf{SG}\,a ::= \mathsf{S}\,(\mathsf{G}\,a)$$

The first, but not the second, gives rise to a monad:

$$\eta \quad :: \; a \to \mathsf{GS}\,a \qquad\qquad \gg\!= \quad :: \; \mathsf{GS}\,a \to (a \to \mathsf{GS}\,b) \to \mathsf{GS}\,b$$
$$\eta\,x := \lambda g.\,\{x\}$$

Finding a composite monad

Two possibilities for layering assignments and alternatives:

$$\mathsf{GS}\,a ::= \mathsf{G}\,(\mathsf{S}\,a) \qquad \mathsf{SG}\,a ::= \mathsf{S}\,(\mathsf{G}\,a)$$

The first, but not the second, gives rise to a monad:

$$\begin{array}{ll} \eta \;\; :: \; a \to \mathsf{GS}\,a & \gg\!= \;\; :: \; \mathsf{GS}\,a \to (a \to \mathsf{GS}\,b) \to \mathsf{GS}\,b \\ \eta\,x := \lambda g.\,\{x\} & m \gg\!= f := \lambda g.\,\bigcup_{x \in m\,g} f\,x\,g \end{array}$$

Getting dynamic

Is GS dynamic?

Getting dynamic

Is GS dynamic? How would we go about making GS dynamic?

What is GS missing, relative to dynamic semantics?

Getting dynamic

Is GS dynamic? How would we go about making GS dynamic?

What is GS missing, relative to dynamic semantics?

- It doesn't treat assignments *as information*!

First, decide on a type

Here's our type for static assignemnts with alternatives:

$$\mathsf{GS}\,a ::= \mathsf{g} \to \mathsf{S}\,a$$

What would a corresponding dynamic version look like? Remember that the hallmark of dynamic systems is the ability to **store assignments**. . .

$$\mathsf{D}\,a ::=$$

First, decide on a type

Here's our type for static assignemnts with alternatives:

$$\mathsf{GS}\,a ::= \mathsf{g} \to \mathsf{S}\,a$$

What would a corresponding dynamic version look like? Remember that the hallmark of dynamic systems is the ability to **store assignments**...

$$\mathsf{D}\,a ::= g \to \mathsf{S}\,(a \times g)$$

Then find a η...

Let's use our static ($\mathsf{GS}\,a ::= \mathsf{g} \to \mathsf{S}\,a$) η as a starting point:

$$\eta\, x := \lambda g.\{x\}$$

Then find a η...

Let's use our static ($\mathsf{GS}\,a ::= \mathsf{g} \to \mathsf{S}\,a$) η as a starting point:

$$\eta\, x := \lambda g.\{x\}$$

How would we have to modify it for $\mathsf{D}\,a ::= \mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$?

$$\eta\, x :=$$

Then find a η...

Let's use our static ($\mathsf{GS}\,a ::= \mathsf{g} \to \mathsf{S}\,a$) η as a starting point:

$$\eta\, x := \lambda g.\{x\}$$

How would we have to modify it for $\mathsf{D}\,a ::= \mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$?

$$\eta\, x := \lambda g.$$

Then find a η...

Let's use our static ($\mathsf{GS}\,a ::= \mathsf{g} \to \mathsf{S}\,a$) η as a starting point:

$$\eta\, x := \lambda g.\{x\}$$

How would we have to modify it for $\mathsf{D}\,a ::= \mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$?

$$\eta\, x := \lambda g.\{(x, g)\}$$

And a $\gg=\ldots$

Let's use our static $(\mathsf{GS}\,a ::= \mathsf{g} \to \mathsf{S}\,a) \gg=$ as a starting point:

$$m \gg= f := \lambda g.\bigcup_{x \in m\,g} f\,x\,g$$

And a $\gg=$...

Let's use our static ($\mathsf{GS}\,a ::= \mathsf{g} \to \mathsf{S}\,a$) $\gg=$ as a starting point:

$$m \gg= f := \lambda g.\, \bigcup_{x \in m\,g} f\,x\,g$$

How would we have to modify it for $\mathsf{D}\,a ::= \mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$?

$$m \gg= f :=$$

And a $\gg=$...

Let's use our static ($\mathsf{GS}\,a ::= \mathsf{g} \to \mathsf{S}\,a$) $\gg=$ as a starting point:

$$m \gg= f := \lambda g. \bigcup_{x \in m\,g} f\,x\,g$$

How would we have to modify it for $\mathsf{D}\,a ::= \mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$?

$$m \gg= f := \lambda g.$$

And a $\gg\!=\ldots$

Let's use our static ($\mathsf{GS}\,a ::= \mathsf{g} \to \mathsf{S}\,a$) $\gg\!=$ as a starting point:

$$m \gg\!= f := \lambda g.\bigcup_{x \in m\,g} f\,x\,g$$

How would we have to modify it for $\mathsf{D}\,a ::= \mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$?

$$m \gg\!= f := \lambda g.\bigcup$$

And a $\gg\!=\ldots$

Let's use our static ($\mathsf{GS}\,a ::= \mathsf{g} \to \mathsf{S}\,a$) $\gg\!=$ as a starting point:

$$m \gg\!= f := \lambda g.\bigcup_{x \in m\,g} f\,x\,g$$

How would we have to modify it for $\mathsf{D}\,a ::= \mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$?

$$m \gg\!= f := \lambda g.\bigcup_{(x,h) \in m\,g}$$

And a $\gg\!=\ldots$

Let's use our static ($\mathsf{GS}\,a ::= \mathsf{g} \to \mathsf{S}\,a$) $\gg\!=$ as a starting point:

$$m \gg\!= f := \lambda g.\bigcup_{x \in m\,g} f\,x\,g$$

How would we have to modify it for $\mathsf{D}\,a ::= \mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$?

$$m \gg\!= f := \lambda g.\bigcup_{(x,h) \in m\,g} f\,x$$

And a $\gg=$...

Let's use our static ($\mathsf{GS}\,a ::= \mathsf{g} \to \mathsf{S}\,a$) $\gg=$ as a starting point:

$$m \gg= f := \lambda g.\, \bigcup_{x \in m\,g} f\,x\,g$$

How would we have to modify it for $\mathsf{D}\,a ::= \mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$?

$$m \gg= f := \lambda g.\, \bigcup_{(x,h) \in m\,g} f\,x\,h$$

Summed up

Fixing a type constructor:

$$\mathsf{D}\,a ::= \mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$$

Along with the associated η and $\gg\!=$ operations:

$$\eta \quad :: \; a \to \mathsf{D}\,a$$

Summed up

Fixing a type constructor:

$$\mathsf{D}\,a ::= \mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$$

Along with the associated η and $\gg=$ operations:

$$\begin{aligned} \eta &:: a \to \mathsf{D}\,a \\ \eta\,x &:= \lambda g.\,\{(x, g)\} \end{aligned}$$

Summed up

Fixing a type constructor:

$$\mathsf{D}\,a ::= \mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$$

Along with the associated η and $\gg\!=$ operations:

$$\eta \quad :: a \to \mathsf{D}\,a \qquad\qquad \gg\!= \quad :: \mathsf{D}\,a \to (a \to \mathsf{D}\,b) \to \mathsf{D}\,b$$
$$\eta\,x := \lambda g.\,\{(x, g)\}$$

Summed up

Fixing a type constructor:

$$\mathsf{D}\,a ::= \mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$$

Along with the associated η and $\gg\!=$ operations:

$$\begin{array}{ll} \eta \;\; :: \; a \to \mathsf{D}\,a & \qquad \gg\!= \;\; :: \; \mathsf{D}\,a \to (a \to \mathsf{D}\,b) \to \mathsf{D}\,b \\ \eta\,x := \lambda g.\,\{(x, g)\} & \qquad m \gg\!= f := \lambda g.\,\bigcup_{(x,h) \in m\,g} f\,x\,h \end{array}$$

Indefinites and pronouns

Bootstrapping a dynamic entry for the indefinite:

$$\mathbf{a}.\mathbf{ling}_{\mathsf{GS}} := \lambda g.\{x \mid \mathbf{ling}\, x\} \qquad \mathbf{a}.\mathbf{ling}_{\mathsf{D}} :=$$

Indefinites and pronouns

Bootstrapping a dynamic entry for the indefinite:

$$\mathbf{a.ling}_{\mathsf{GS}} := \lambda g.\{x \mid \mathbf{ling}\, x\} \qquad \mathbf{a.ling}_{\mathsf{D}} := \lambda g.\{(x, g) \mid \mathbf{ling}\, x\}$$

Bootstrapping a dynamic entry for pronouns:

$$\mathbf{she_0}_{\mathsf{GS}} := \lambda g.\{g_0\} \qquad \mathbf{she_0}_{\mathsf{D}} :=$$

Indefinites and pronouns

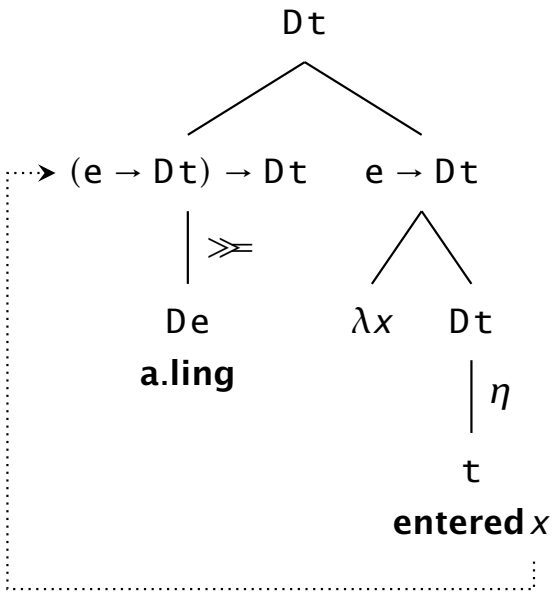
Bootstrapping a dynamic entry for the indefinite:

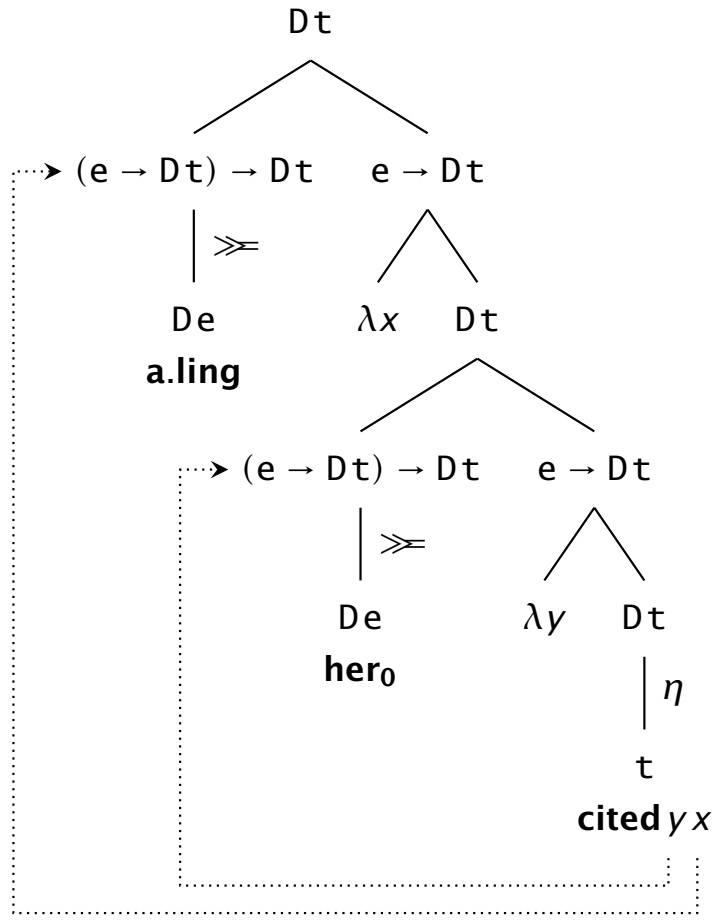
$$\mathbf{a.ling}_{\mathsf{GS}} := \lambda g.\{x \mid \mathbf{ling}\, x\} \qquad \mathbf{a.ling}_{\mathsf{D}} := \lambda g.\{(x, g) \mid \mathbf{ling}\, x\}$$

Bootstrapping a dynamic entry for pronouns:

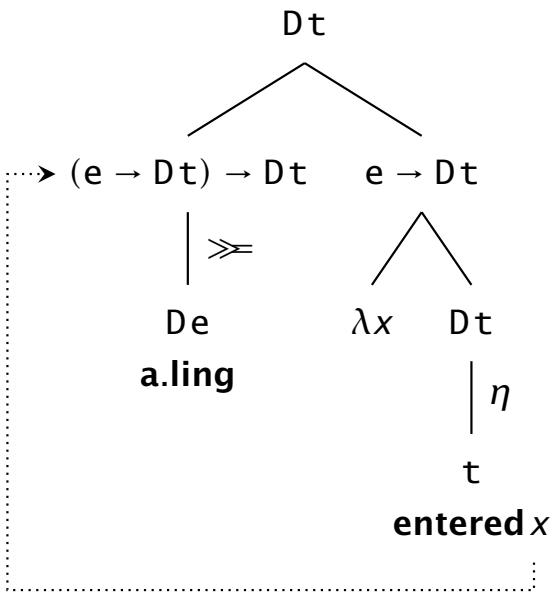
$$\mathbf{she_0}_{\mathsf{GS}} := \lambda g.\{g_0\} \qquad \mathbf{she_0}_{\mathsf{D}} := \lambda g.\{(g_0, g)\}$$

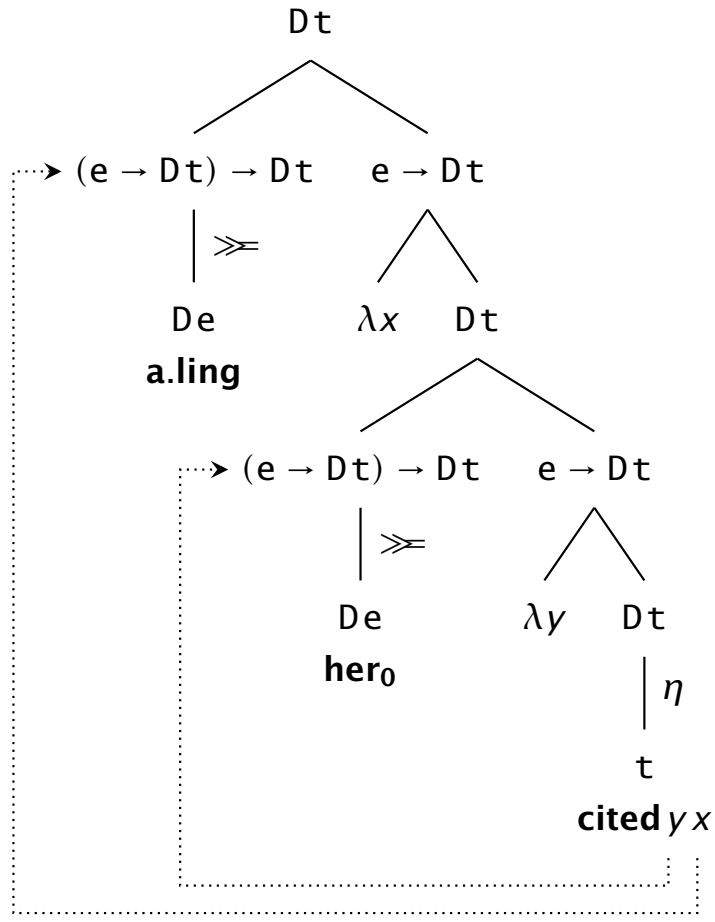
Some simple derivations





Some simple derivations





$= \lambda g.\{(\textbf{entered}\, x, g) \mid \textbf{ling}\, x\}$

$= \lambda g.\{(\textbf{cited}\, g_0\, x, g) \mid \textbf{ling}\, x\}$

Effecting binding?

Our old operator for effecting binding (cf. Büring 2005):

$$\beta^n :: (a \to \mathsf{g} \to b) \to a \to \mathsf{g} \to b$$

Effecting binding?

Our old operator for effecting binding (cf. Büring 2005):

$$\beta^n :: (a \to \mathsf{g} \to b) \to a \to \mathsf{g} \to b$$
$$\beta^n := \lambda f.\lambda x.\lambda g.f\, x\, g^{n \to x}$$

Almost an identity function, but for the fact that it changes the assignment that's fed to f.

Do we need a new one in the dynamic setting?

Effecting binding?

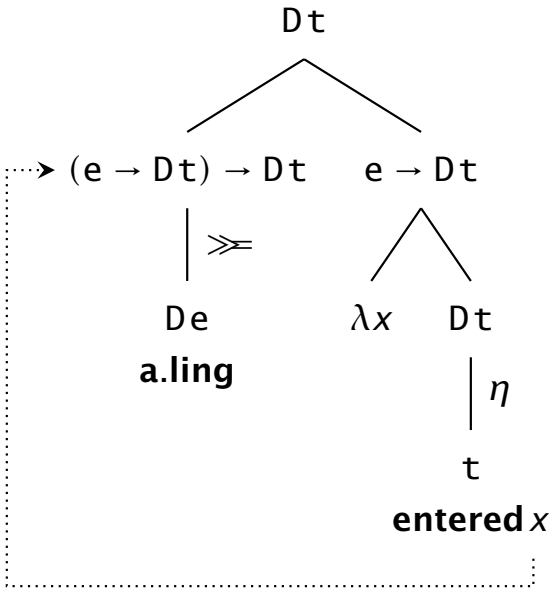
Our old operator for effecting binding (cf. Büring 2005):

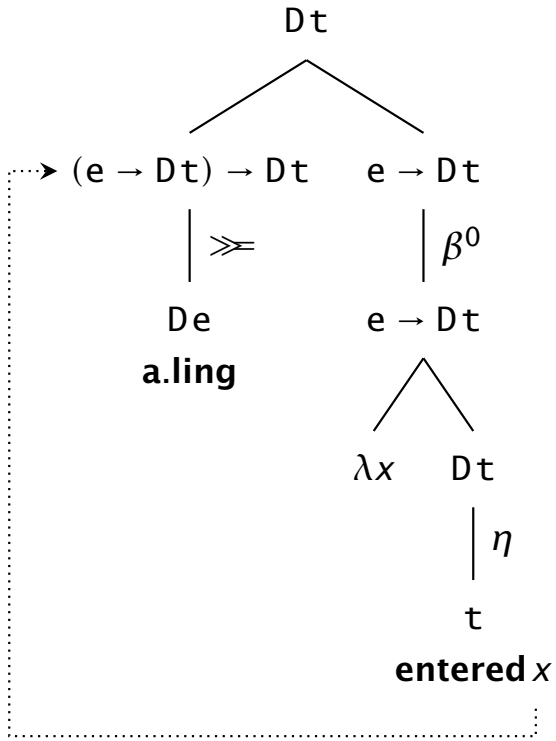
$$\beta^n :: (a \to \mathsf{g} \to b) \to a \to \mathsf{g} \to b$$
$$\beta^n := \lambda f.\lambda x.\lambda g.f\,x\,g^{n \to x}$$

Almost an identity function, but for the fact that it changes the assignment that's fed to f.

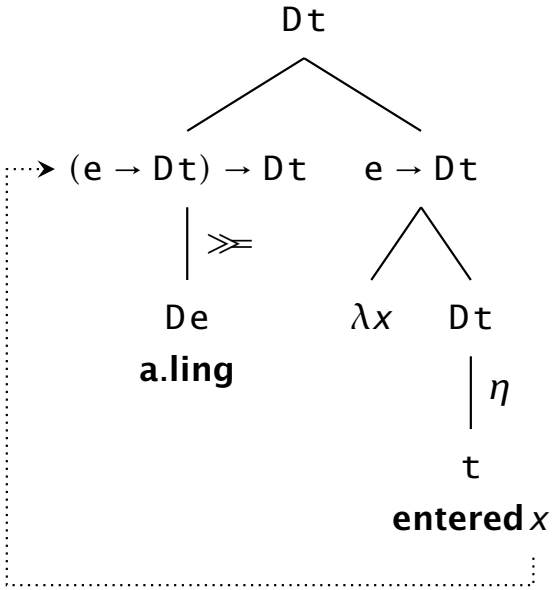
Do we need a new one in the dynamic setting? *No!* All that β^n requires is some assignment-sensitivity. That's true of G, GS, and D!

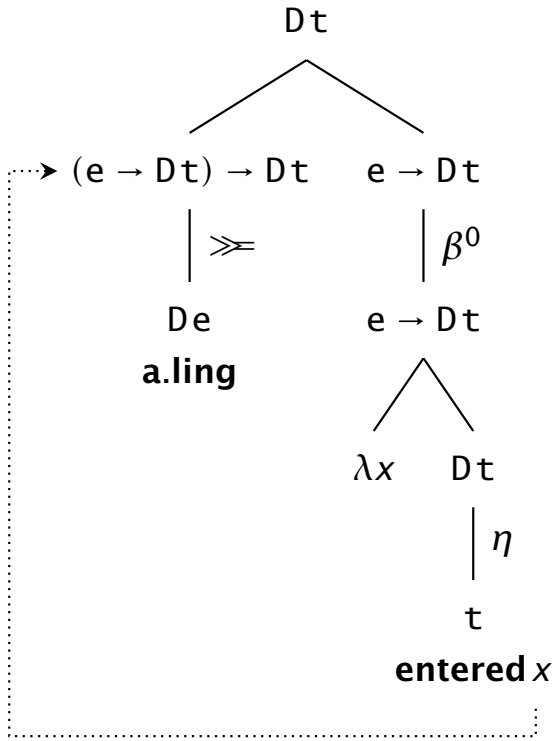
Basic derivations





Basic derivations





$= \lambda g.\{(\textbf{entered}\, x, g) \mid \textbf{ling}\, x\}$

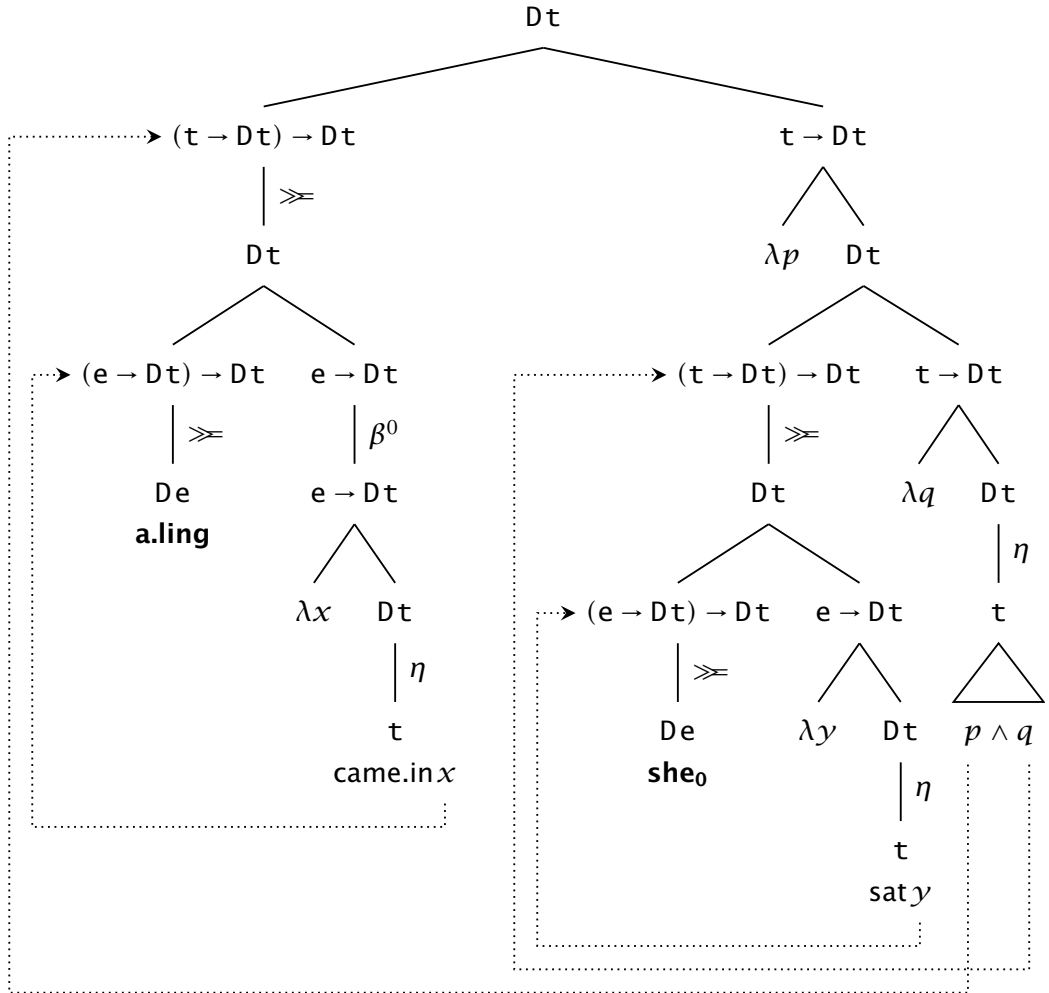
$= \lambda g.\{(\textbf{entered}\, x, g^{0 \to x}) \mid \textbf{ling}\, x\}$

And that's the key

Because D characterizes things that *store* information about the assignment, β^n operators introduce assignment shifts that live on.

That unlocks dynamic binding, just as in the non-monadic setting.

A derivation of cross-sentential binding



$$= \lambda g.\{(\textbf{came.in}\, x \wedge \textbf{sat}\, x, g^{0 \to x}) \mid \textbf{ling}\, x\}$$

Remarks on this derivation

Notice *how similar* it is to exceptional scope derivations.

- Something takes scope at the edge of the island, and then the island itself takes scope.
- This has the effect of *transmitting* the 'effects' in the island into the constituent over which the island scoped.

In static systems, this gave us exceptional 'quantificational scope'.

In a dynamic system, this **also** gives us exceptional 'binding scope'.

Alternatives, with static and dynamic binding

$T\,a$	$\eta\,x$	$m \gg\!= f$	**a.ling**	**she**$_0$
$\mathsf{S}\,a$	$\{x\}$	$\bigcup_{x \in m} f\,x$	$\{x \mid \mathbf{ling}\,x\}$	—
$\mathsf{g} \to \mathsf{S}\,a$	$\lambda g.\,\{x\}$	$\lambda g.\,\bigcup_{x \in mg} f\,x\,g$	$\lambda g.\,\{x \mid \mathbf{ling}\,x\}$	$\lambda g.\,\{g_0\}$
$\mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$	$\lambda g.\,\{(x, g)\}$	$\lambda g.\,\bigcup_{(x,h) \in mg} f\,x\,h$	$\lambda g.\,\{(x, g) \mid \mathbf{ling}\,x\}$	$\lambda g.\,\{(g_0, g)\}$

On extending a theory

The dynamic system with alternatives is a **strict extension** of the static system with alternatives. In particular, if we got rid of β^n operators, the two systems would be *equivalent*.

This means that all of the initial results in the static semantics — island-insensitivity for alternative generators, selectivity outside islands, and so on — are *preserved* in the dynamic system.

The State monad

Factoring something out

Recall that our GS monad (static assignments with alternatives) was in some sense a *composition* of G and S: $\mathsf{GS}\,a ::= \mathsf{G}\,(\mathsf{S}\,a)$.

So it would be interesting to consider whether the D monad (dynamic assignments with alternatives) could be broken down in a similar way, cleaving off alternatives from the apparatus underlying dynamic binding.

Up from statics

Our monad for static binding (*sans* alternatives):

$$\eta \quad :: a \to \mathsf{G}\,a \qquad\qquad \gg\!= \quad :: \mathsf{G}\,a \to (a \to \mathsf{G}\,b) \to \mathsf{G}\,b$$
$$\eta\,x := \lambda g.\,x \qquad\qquad m \gg\!= f := \lambda g.\,f\,(m\,g)\,g$$

How might a dynamic version look? First, a type constructor:

$$\mathsf{H}\,a ::=$$

Up from statics

Our monad for static binding (*sans* alternatives):

$$\eta \quad :: a \to \mathsf{G}\,a \qquad\qquad \gg\!= \quad :: \mathsf{G}\,a \to (a \to \mathsf{G}\,b) \to \mathsf{G}\,b$$
$$\eta\,x := \lambda g.\,x \qquad\qquad m \gg\!= f := \lambda g.\,f\,(m\,g)\,g$$

How might a dynamic version look? First, a type constructor:

$$\mathsf{H}\,a ::= \mathsf{g} \to (a, \mathsf{g})$$

Then find a η and a $\gg\!=$ that make sense:

Up from statics

Our monad for static binding (*sans* alternatives):

$$\eta \quad :: a \to \mathsf{G}\,a \qquad\qquad \gg\!= \quad :: \mathsf{G}\,a \to (a \to \mathsf{G}\,b) \to \mathsf{G}\,b$$
$$\eta\,x := \lambda g.\,x \qquad\qquad m \gg\!= f := \lambda g.\,f\,(m\,g)\,g$$

How might a dynamic version look? First, a type constructor:

$$\mathsf{H}\,a ::= \mathsf{g} \to (a, \mathsf{g})$$

Then find a η and a $\gg\!=$ that make sense:

$$\eta \quad :: a \to \mathsf{H}\,a$$
$$\eta\,x :=$$

Up from statics

Our monad for static binding (*sans* alternatives):

$$\begin{array}{ll} \eta \quad :: a \to \mathsf{G}\,a & \gg\!= \quad :: \mathsf{G}\,a \to (a \to \mathsf{G}\,b) \to \mathsf{G}\,b \\ \eta\,x := \lambda g.\,x & m \gg\!= f := \lambda g.\,f\,(m\,g)\,g \end{array}$$

How might a dynamic version look? First, a type constructor:

$$\mathsf{H}\,a ::= \mathsf{g} \to (a, \mathsf{g})$$

Then find a η and a $\gg\!=$ that make sense:

$$\begin{array}{ll} \eta \quad :: a \to \mathsf{H}\,a & \gg\!= \quad :: \mathsf{H}\,a \to (a \to \mathsf{H}\,b) \to \mathsf{H}\,b \\ \eta\,x := \lambda g.\,(x, g) & m \gg\!= f := \end{array}$$

Up from statics

Our monad for static binding (*sans* alternatives):

$$\begin{array}{ll} \eta \quad :: a \to \mathsf{G}\,a & \gg\!= \quad :: \mathsf{G}\,a \to (a \to \mathsf{G}\,b) \to \mathsf{G}\,b \\ \eta\,x := \lambda g.\,x & m \gg\!= f := \lambda g.\,f\,(m\,g)\,g \end{array}$$

How might a dynamic version look? First, a type constructor:

$$\mathsf{H}\,a ::= \mathsf{g} \to (a, \mathsf{g})$$

Then find a η and a $\gg\!=$ that make sense:

$$\begin{array}{ll} \eta \quad :: a \to \mathsf{H}\,a & \gg\!= \quad :: \mathsf{H}\,a \to (a \to \mathsf{H}\,b) \to \mathsf{H}\,b \\ \eta\,x := \lambda g.\,(x, g) & m \gg\!= f := \lambda g.\,f\,(m\,g)_{fst}\,(m\,g)_{snd} \end{array}$$

Is it a monad?

Indeed! It's commonly known as the **State monad**.

$$\eta \quad :: a \to \mathsf{H}\, a \qquad\qquad \gg\!= \quad :: \mathsf{H}\, a \to (a \to \mathsf{H}\, b) \to \mathsf{H}\, b$$
$$\eta\, x := \lambda g.\, (x, g) \qquad\qquad m \gg\!= f := \lambda g.\, f\, (m\, g)_{fst}\, (m\, g)_{snd}$$

Let's check that Left ID is satisfied:

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

Is it a monad?

Indeed! It's commonly known as the **State monad**.

$$\begin{array}{ll} \eta \;\; :: \; a \to \mathsf{H}\, a & \gg\!= \;\; :: \; \mathsf{H}\, a \to (a \to \mathsf{H}\, b) \to \mathsf{H}\, b \\ \eta\, x := \lambda g.\, (x, g) & m \gg\!= f := \lambda g.\, f\, (m\, g)_{fst}\, (m\, g)_{snd} \end{array}$$

Let's check that Left ID is satisfied:

$$\eta\, x \gg\!= f = (\lambda g.\, (x, g)) \gg\!= f$$

Is it a monad?

Indeed! It's commonly known as the **State monad**.

$$\begin{aligned} \eta &:: a \to \mathsf{H}\,a & \gg\!= &:: \mathsf{H}\,a \to (a \to \mathsf{H}\,b) \to \mathsf{H}\,b \\ \eta\,x &:= \lambda g.\,(x, g) & m \gg\!= f &:= \lambda g.\,f\,(m\,g)_{fst}\,(m\,g)_{snd} \end{aligned}$$

Let's check that Left ID is satisfied:

$$\begin{aligned} \eta\,x \gg\!= f &= (\lambda g.\,(x, g)) \gg\!= f \\ &= \lambda g.\,f\,(x, g)_{fst}\,(x, g)_{snd} \end{aligned}$$

Is it a monad?

Indeed! It's commonly known as the **State monad**.

$$\begin{aligned} \eta &:: a \to \mathsf{H}\,a & \gg\!= &:: \mathsf{H}\,a \to (a \to \mathsf{H}\,b) \to \mathsf{H}\,b \\ \eta\,x &:= \lambda g.\,(x, g) & m \gg\!= f &:= \lambda g.\,f\,(m\,g)_{fst}\,(m\,g)_{snd} \end{aligned}$$

Let's check that Left ID is satisfied:

$$\begin{aligned} \eta\,x \gg\!= f &= (\lambda g.\,(x, g)) \gg\!= f \\ &= \lambda g.\,f\,(x, g)_{fst}\,(x, g)_{snd} \\ &= \lambda g.\,f\,x\,g \end{aligned}$$

Is it a monad?

Indeed! It's commonly known as the **State monad**.

$$\begin{aligned} \eta &:: a \to \mathsf{H}\,a & \gg\!= &:: \mathsf{H}\,a \to (a \to \mathsf{H}\,b) \to \mathsf{H}\,b \\ \eta\,x &:= \lambda g.\,(x, g) & m \gg\!= f &:= \lambda g.\,f\,(m\,g)_{fst}\,(m\,g)_{snd} \end{aligned}$$

Let's check that Left ID is satisfied:

$$\begin{aligned} \eta\,x \gg\!= f &= (\lambda g.\,(x, g)) \gg\!= f \\ &= \lambda g.\,f\,(x, g)_{fst}\,(x, g)_{snd} \\ &= \lambda g.\,f\,x\,g \\ &= f\,x \qquad\qquad \square \end{aligned}$$

“Mutable state”

Programmers use the State monad for the same kinds of stuff as us!

Specifically, the State monad is used to talk about “mutable state”, whereby variable assignments can be propagated to distant lands:

```
x = 0 ;  // assign x to 0
x     ;  // evaluates to 0
x++   ;  // increment x
x     ;  // evaluates to 1
// arbitrarily many lines w/o re-assigning x
x     ;  // evaluates to 1
```

A disanalogy

Whereas GS really is a composition of G and S (if you're interested in the gory details, check it out here), i.e., $\mathsf{GS} ::= \mathsf{G}\,(\mathsf{S}\,a)\ldots$

The same **cannot** be said for D! It isn't a composition of H and S:

$$\mathsf{H}\,(\mathsf{S}\,a) = \mathsf{g} \to (\mathsf{S}\,a \times \mathsf{g}) \qquad \mathsf{D}\,a ::= \mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$$

So there must be another way to "compose" monads, other than by strictly composing them (Liang, Hudak & Jones 1995).

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

Charlow, Simon. 2016. Post-suppositions and semantic theory. Unpublished ms.

Dekker, Paul. 1994. Predicate logic with anaphora. In Mandy Harvey & Lynn Santelmann (eds.), *Proceedings of Semantics and Linguistic Theory 4*, 79–95. Ithaca, NY: Cornell University. http://dx.doi.org/10.3765/salt.v4i0.2459.

Liang, Sheng, Paul Hudak & Mark Jones. 1995. Monad transformers and modular interpreters. In *22nd ACM Symposium on Principles of Programming Languages (POPL '95)*, 333–343. ACM Press.