Dynamic alternatives II

Computational semantics seminar

March 1, 2017

Dynamic semantics

The basic picture: updating the context

$$\begin{tikzcd}
g \arrow[r, dotted] & \llbracket \text{Polly}^{n} \text{ left} \rrbracket \arrow[r, dotted] & g^{n \to \mathbf{p}} \arrow[r, dotted] & \llbracket \text{she}_{n} \text{ was tired} \rrbracket \arrow[r, dotted] & g^{n \to \mathbf{p}}
\end{tikzcd}$$

$$\begin{tikzcd}
 & & g^{n \to \mathbf{a}} \arrow[r, dotted] & \llbracket \text{she}_{n} \text{ was tired} \rrbracket \arrow[r, dotted] & g^{n \to \mathbf{a}} \\
g \arrow[r, dotted] & \llbracket \text{a ling}^{n} \text{ left} \rrbracket \arrow[ur, dotted] \arrow[r, dotted] \arrow[dr, dotted] \arrow[ddr, dotted] & g^{n \to \mathbf{b}} \arrow[r, dotted] & \llbracket \text{she}_{n} \text{ was tired} \rrbracket \arrow[r, dotted] & {} \\
 & & g^{n \to \mathbf{c}} \arrow[r, dotted] & \llbracket \text{she}_{n} \text{ was tired} \rrbracket \arrow[r, dotted] & {} \\
 & & g^{n \to \mathbf{d}} \arrow[r, dotted] & \llbracket \text{she}_{n} \text{ was tired} \rrbracket \arrow[r, dotted] & g^{n \to \mathbf{d}}
\end{tikzcd}$$

Point-wise and update-theoretic perspectives on dynamics

There's a couple way to write down a semantics with this behavior. They're basically equivalent for present purposes.

Standard "point-wise" dynamic semantics treats sentence meanings as **relations on contexts** (with contexts modeled as assignment functions):

$$g \to Sg$$

Update semantics treats sentence meanings as **functions from contexts into contexts** (with contexts modeled as *sets* of assignments):

$$Sg \to Sg$$

Eyes on the prize

What're the key features of both $g \to Sg$ and $Sg \to Sg$?

Eyes on the prize

What're the key features of both $g \to Sg$ and $Sg \to Sg$?

I see two important ones:

- They **store** output assignments.
- They potentially do so **nondeterministically** ($>$1 output returned).

Key pieces of dynamics (from point-wise perspective)

Sentences **update the context** (assignment function).

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

Sentences with indefinites do so **nondeterministically**.

$$[\![\text{a ling}^n \text{ left}]\!] = \lambda g.\,\{g^{n\to x} \mid \mathbf{ling}\,x,\ \mathbf{left}\,x\}$$

Dynamic conjunction passes updated contexts between sentences.

$$L\,;R = \lambda g.\bigcup_{h\in L g} R\,h$$

An example

Applying dynamic conjunction to a sentence with an indefinite:

$$[\![\text{a ling}^{n} \text{ left and } X]\!] = \lambda g. \bigcup_{h \in \{g^{n \to x} \mid \mathbf{ling}\, x, \mathbf{left}\, x\}} [\![X]\!]\, h$$

The modified assignments are fed directly to $[\![X]\!]$!

More generally, this turns on the associativity of dynamic conjunction:

$$(\textit{indef} \,;\, \textit{predication}) \,;\, \textit{etc} = \textit{indef} \,;\, (\textit{predication} \,;\, \textit{etc})$$

Some "meta"-semantics

Sentences — things we associate with truth values or facts — are the only things it makes sense to associate with type g → Sg.

$$\phi\, g = \{\,\} \iff \phi \text{ is } \textbf{false} \text{ at } g$$
$$\phi\, g \neq \{\,\} \iff \phi \text{ is } \textbf{true} \text{ at } g$$

To capture *sub-sentential* dynamic effects, all denotations are "lifted" into higher-order functions that operate on sentence-sized constituents:

$$[\![\text{Polly}^n]\!] = \lambda f.\lambda g.\, c\, \mathbf{p}\, g^{n\to\mathbf{p}} \qquad \text{type: } (\mathrm{e} \to \mathrm{g} \to \mathrm{S\,g}) \to \mathrm{g} \to \mathrm{S\,g}$$

$$[\![\text{she}_n]\!] = \lambda f.\lambda g.\, f\, g_n\, g \qquad \text{type: } (\mathrm{e} \to \mathrm{g} \to \mathrm{S\,g}) \to \mathrm{g} \to \mathrm{S\,g}$$

$$[\![\text{a linguist}^n]\!] = \lambda f.\lambda g. \bigcup_{x \in \mathbf{ling}} f\, x\, g^{n\to x} \qquad \text{type: } (\mathrm{e} \to \mathrm{g} \to \mathrm{S\,g}) \to \mathrm{g} \to \mathrm{S\,g}$$

Monadic dynamic semantics

What’s in a monad

$$\begin{tikzcd}
a \arrow[r, "\mathbf{LIFT}_{T\,b}"] \arrow[d, "\eta"] & (a \to T\,b) \to T\,b \\
T\,a \arrow[ur, "\gg="] &
\end{tikzcd}$$

The diagram should commute, and the following should hold:

- **Right identity:** $m \gg= \eta = m$
- **Associativity:** $(m \gg= \lambda x.f\,x) \gg= g = m \gg= (\lambda x.f\,x \gg= g)$

Is dynamic semantics monadic? Points (redux):

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 dynamic semantics monadic? Counterpoints (redux):

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!

Up from statics

$$\begin{tikzcd}
 & a \arrow[dl] \arrow[dr] & \\
\mathrm{g} \to a \arrow[dr] & & \mathrm{S}\,a \arrow[dl] \\
 & \mathrm{g} \to \mathrm{S}\,a \arrow[d] & \\
 & ?????????? &
\end{tikzcd}$$

Up from statics

$$\begin{tikzcd}
 & a \arrow[dl] \arrow[dr] & \\
\mathsf{g} \to a \arrow[dr] & & \mathsf{S}\,a \arrow[dl] \\
 & \mathsf{g} \to \mathsf{S}\,a \arrow[d] & \\
 & \mathsf{g} \to \mathsf{S}\,(a \times g) &
\end{tikzcd}$$

Two perspectives on assignments and alternatives

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

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

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

$$\begin{aligned} \eta &:: a \to \mathsf{D}\,a \\ \eta\,x &:= \end{aligned}$$

Two perspectives on assignments and alternatives

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

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

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

$$\begin{aligned} \eta &:: a \to \mathsf{D}\,a & \gg\!= &:: \mathsf{D}\,a \to (a \to \mathsf{D}\,b) \to \mathsf{D}\,b \\ \eta\,x &:= \lambda g.\,\{(x, {\color{red}g})\} & m \gg\!= f &:= \end{aligned}$$

Two perspectives on assignments and alternatives

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

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

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

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

Indefinites and pronouns

Bootstrapping a dynamic entry for the indefinite:

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

Indefinites and pronouns

Bootstrapping a dynamic entry for the indefinite:

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

Bootstrapping a dynamic entry for pronouns:

$$\textbf{she}_{\mathbf{0}\text{GS}} := \lambda g.\, \{g_0\} \qquad \textbf{she}_{\mathbf{0}\text{D}} :=$$

Indefinites and pronouns

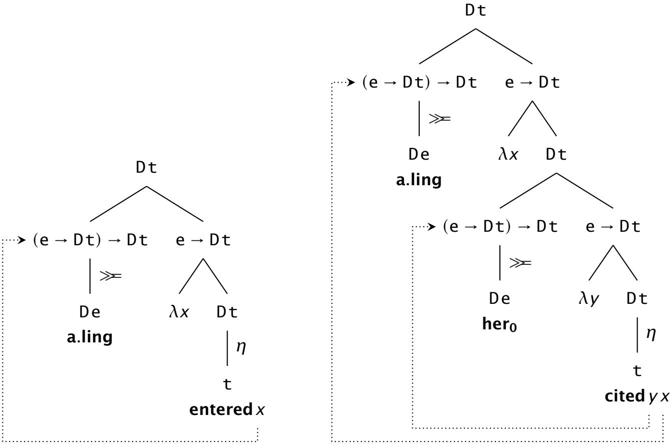
Bootstrapping a dynamic entry for the indefinite:

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

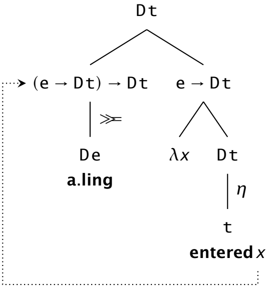
Bootstrapping a dynamic entry for pronouns:

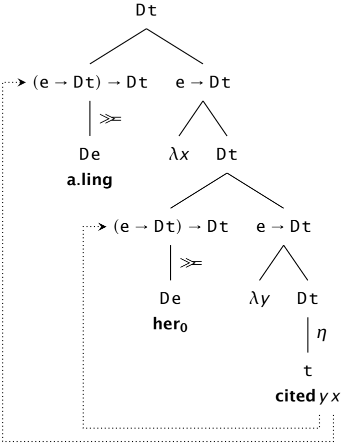
$$\textbf{she}_{\mathbf{0}\text{GS}} := \lambda g.\, \{g_0\} \qquad \textbf{she}_{\mathbf{0}\text{D}} := \lambda g.\, \{(g_0, g)\}$$

Some simple derivations



Some simple derivations





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

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):

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

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?

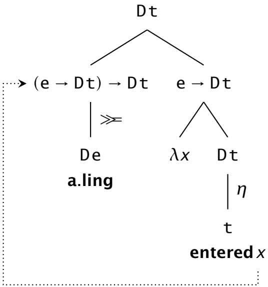
Our old operator for effecting binding (cf. Büring 2005):

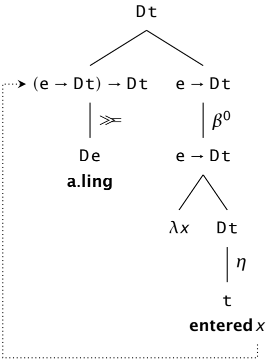
$$\begin{aligned} \beta^n &:: (a \to \mathsf{g} \to b) \to a \to \mathsf{g} \to b \\ \beta^n &:= \lambda f.\lambda x.\lambda g.f\,x\,g^{n\to x} \end{aligned}$$

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

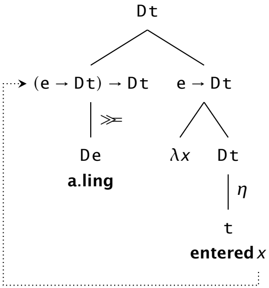
Do we need a new one in the dynamic setting? *No!* All that β^n requires is some assignment-sensitivity. That's true of G, GS, and D!

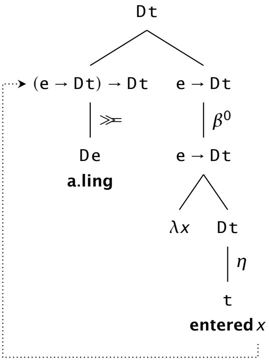
Basic derivations with β





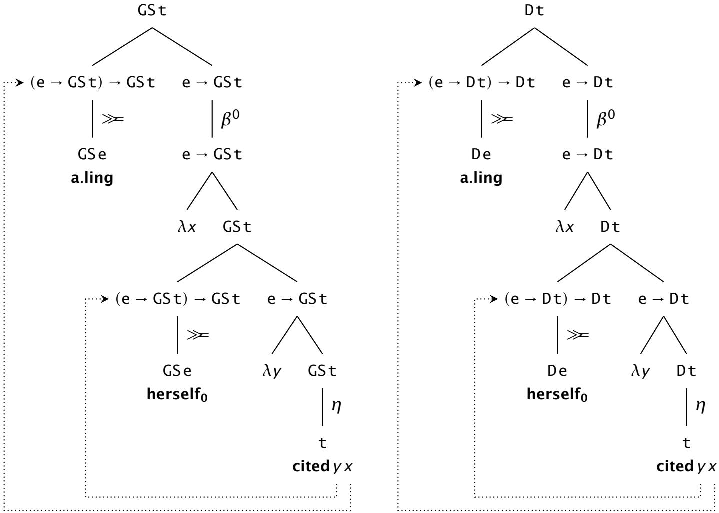
Basic derivations with β



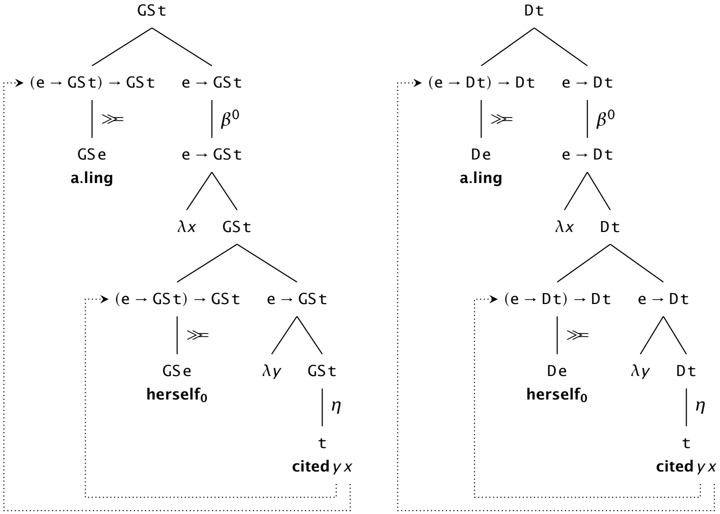


$$= \lambda g.\,\{(\textbf{entered}\,x,\ g) \mid \textbf{ling}\,x\} \qquad = \lambda g.\,\{(\textbf{entered}\,x,\ g^{0\to x}) \mid \textbf{ling}\,x\}$$

Binding a pronoun

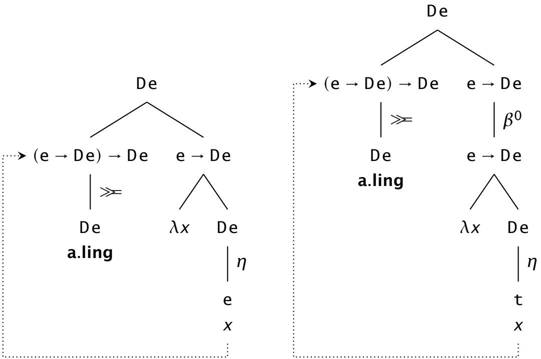


Binding a pronoun

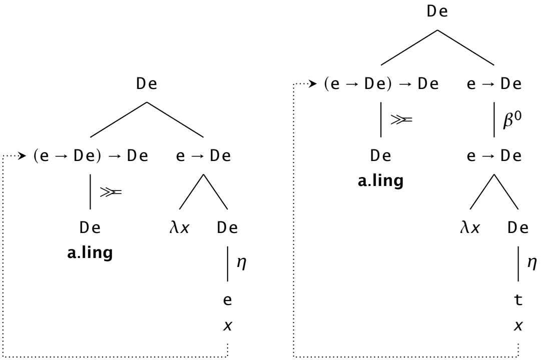


$$= \lambda g.\,\{\textbf{cited}\,x\,x \mid \textbf{ling}\,x\} \qquad = \lambda g.\,\{(\textbf{cited}\,x\,x, g^{0\to x}) \mid \textbf{ling}\,x\}$$

An interesting twist



An interesting twist



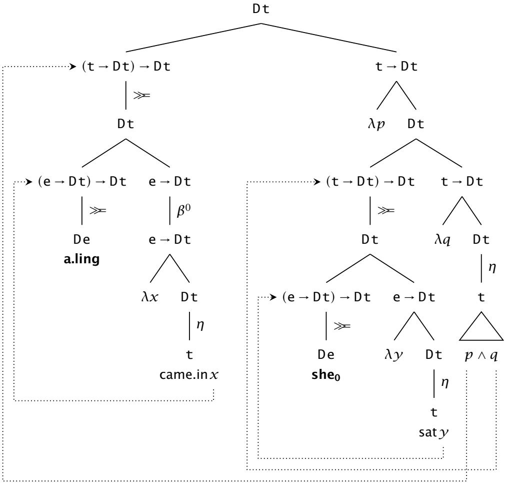
$= \textbf{a.ling}$ $\quad = \lambda g.\, \{(x, g^{0\to x}) \mid \textbf{ling}\, x\}$

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

$T\,a$	$\eta\,x$	$m \ggg= f$	$\mathbf{a.ling}$	$\mathbf{she_0}$
$\mathsf{S}\,a$	$\{x\}$	$\bigcup_{x \in m} f\,x$	$\{x \mid \mathbf{ling}\,x\}$	—
$\mathsf{g} \to \mathsf{S}\,a$	$\lambda g.\,\{x\}$	$\lambda g.\,\bigcup_{x \in mg} f\,x\,g$	$\lambda g.\,\{x \mid \mathbf{ling}\,x\}$	$\lambda g.\,\{g_0\}$
$\mathsf{g} \to \mathsf{S}\,(a \times \mathsf{g})$	$\lambda g.\,\{(x, g)\}$	$\lambda g.\,\bigcup_{(x,h) \in mg} f\,x\,h$	$\lambda g.\,\{(x, g) \mid \mathbf{ling}\,x\}$	$\lambda g.\,\{(g_0, g)\}$

Why?

Exceptional scope

Standard dynamic semantics simply has nothing to say about the exceptional scope behavior of indefinites.

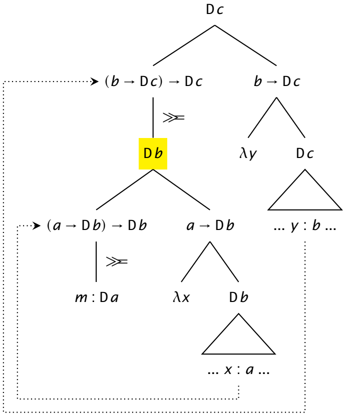
Indefinites are treated as special (they can introduce assignment shifts that survive after a sentence is evaluated, unlike, e.g., universals), but that special-ness doesn’t explain exceptional scope!

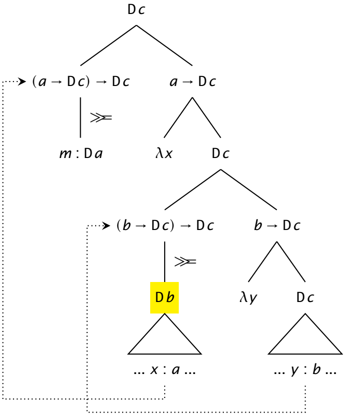
On extending a theory

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*.

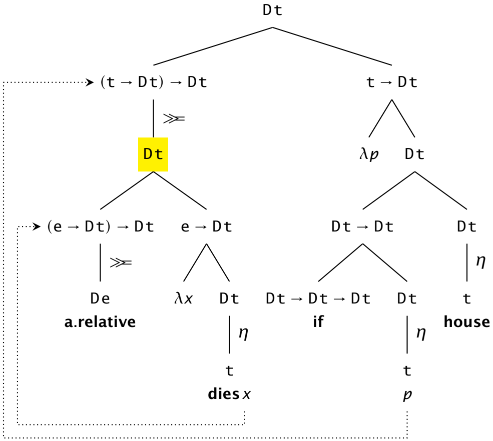
This means that all of the initial results in the static semantics — island-insensitivity for alternative generators, selectivity outside islands, and so on — are *preserved* in the dynamic system.

The associativity law, for D





Island-insensitivity



Standard dynamics falls short of its goals

Does dynamic semantics actually liberate binding from scope? Hardly. Indeed, in many cases it seems like you wouldn't want this!

1. I don't own a car^i. It$_i$'s a VW. $*\neg \gg \exists$
2. If a farmer^i owns a donkey^j, he$_i$ beats it$_j$. It brays. $*\textbf{if} \gg \exists$
3. Everybody owns a car^i. It$_i$'s a VW. $*\forall \gg \exists$

Dynamically closed meanings

Negation requires its propositional argument to be dynamically false, **tossing out** any new binding information accrued in its scope:

$$\textbf{not}\, p := \lambda g. \begin{cases} \{g\} \text{ if } p\,g = \emptyset \\ \{\ \} \text{ otherwise} \end{cases}$$

Similarly for conditionals (cf. $p \Rightarrow q \equiv \neg(p \wedge \neg q)$):

$$\textbf{if}\, p\, q := \textbf{not}\,(p\,;\textbf{not}\, q)$$

And similarly for, e.g., universal quantifiers (cf. $\forall x.\,\phi \equiv \neg\exists x.\,\neg\phi$):

$$\textbf{every}\, f\, g := \textbf{not}\,(\textbf{a}\, f\,(\lambda x.\,\textbf{not}\,(g\,x)))$$

Sloppy readings in VP ellipsis

Pronouns in elided VPs can be understood in different ways than the corresponding pronoun the antecedent VP:

1. John^i likes his$_i$ mom. Bill^j doesn't Δ.

   $\Delta \longrightarrow$ like his$_j$ mom

2. John^i's mom likes him$_i$. Bill^j's mom doesn't Δ.

   $\Delta \longrightarrow$ like him$_j$

On virtually anybody's theory of sloppy readings, this requires the elided pronoun to be semantically **bound** (cf. Rooth 1992, Tomioka 1999).

“Surprising” sloppy readings

Sloppy readings are basically insensitive to how deeply embedded the sloppy pronoun’s antecedent is:

1. If everybody who [hates John^i] comes, I’ll feel bad for him$_i$.
   But if everybody who [hates Jeff^j] comes, I won’t Δ.
   $\Delta \longrightarrow$ feel bad for him$_j$

Even on dynamic pictures of meaning, this behavior is unexpected! Why?

“Surprising” sloppy readings

Sloppy readings are basically insensitive to how deeply embedded the sloppy pronoun’s antecedent is:

1. If everybody who [hates John^i] comes, I’ll feel bad for him$_i$.
   But if everybody who [hates Jeff^j] comes, I won’t Δ.
   $\Delta \longrightarrow$ feel bad for him$_j$

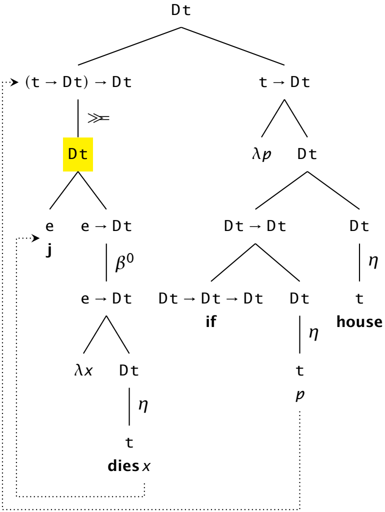
Even on dynamic pictures of meaning, this behavior is unexpected! Why?

- Sloppy pronouns need to be semantically **bound**.
- But in our example, *Jeff* is trapped on an island, under the scope of the dynamically closed *every*!

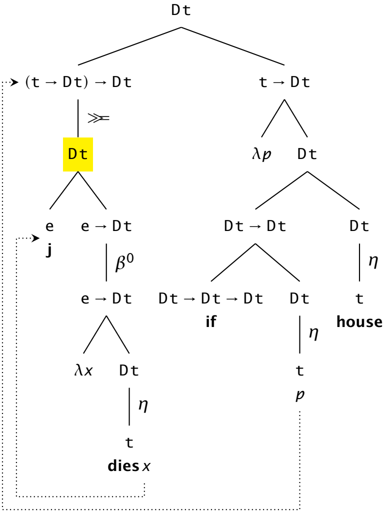
Does this remind you of anything?

This looks an awful lot like **exceptional scope behavior**, but with respect to pure binding.

Exceptional scope feeds binding!



Exceptional scope feeds binding!



$$= \lambda g.\, \{ (\mathbf{dies}\,\mathbf{j} \Rightarrow \mathbf{house}, g^{0 \to \mathbf{j}}) \}$$

What do we have?

Because the monadic character of D guarantees exceptional scope behavior (both for indefiniteness and assignment-updates), we have a *fully* general theory of sloppy readings.

More generally, we've liberated binding from scope (cf. Safir 2004).

The State monad

An interesting discontinuity

Heretofore, our monadic theories haven't been giving us things that are different *in kind* from the sorts of meanings you get in other theories.

- Sets of alternatives
- Assignment-dependent meanings
- Assignment-dependent sets of alternatives

But when we go monadic-dynamic, a new perspective on meaning is forced on us! We aren't simply recapitulating standard dynamic theories. We're doing something a bit more interesting.

It is worth pondering this some more. . .

Factoring something out

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

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):

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

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

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

Up from statics

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

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

How 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):

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

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

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

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

$$\begin{array}{l} \eta \quad :: \; a \to \mathrm{H}\,a \\ \eta\,x := \end{array}$$

Up from statics

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

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

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:

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

Up from statics

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

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

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

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

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

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

Is it a monad?

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

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

Let's check that Left ID is satisfied:

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

Is it a monad?

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

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

Let's check that Left ID is satisfied:

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

Is it a monad?

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

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

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**.

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

Let's check that Left ID is satisfied:

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

Is it a monad?

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

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

Let's check that Left ID is satisfied:

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

“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., 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.

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.

Rooth, Mats. 1992. Ellipsis redundancy and reduction redundancy. In Steven Berman & Arild Hestvik (eds.), *Proceedings of the Stuttgart Workshop on Ellipsis, no. 29 in Arbeitspapiere des SFB 340*. Stuttgart: University of Stuttgart.

Safir, Ken. 2004. *The Syntax of (In)dependence*. Cambridge, MA: MIT Press.

Tomioka, Satoshi. 1999. A sloppy identity puzzle. *Natural Language Semantics* 7(2). 217–241. http://dx.doi.org/10.1023/A:1008309217917.