

Effectful composition in natural language semantics

Adjunctions

Dylan Bumford (UCLA) Simon Charlow (Yale)

NASSLLI 2025 @ UW

June 27, 2025

“Dynamic” semantics

Unexpected anaphora

Indefinites can antecede pronouns they don't (can't) scope over:

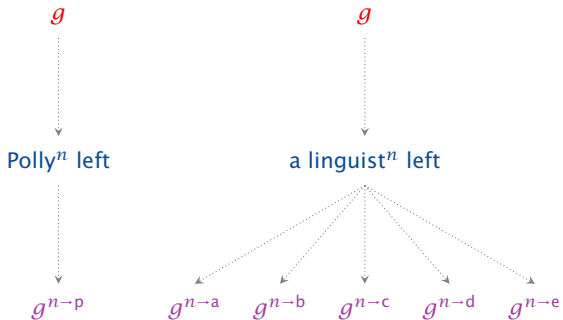
1. Polly_{*i*} walked in the park. She_{*i*} whistled.
2. A linguist_{*i*} walked in the park. She_{*i*} whistled.
3. If a man_{*i*} is from Omaha, he_{*i*} isn't from Lincoln.

The meanings of (2) and (3) aren't captured by the natural translations:

- ▶ $\exists x(Lx \wedge Px) \wedge Wx$
- ▶ $\exists x(Mx \wedge Ox) \Rightarrow \neg Lx$

Dynamic semantics explains (2) and (3) by assimilating them to (1) (Geach 1962, Lewis 1975, Karttunen 1976, Heim 1982, many more).

The basic intuition



Dynamic semantics

Meanings update the context:

$$\blacktriangleright c[F(x_1, \dots, x_n)] := \{g \in c \mid \llbracket F(x_1, \dots, x_n) \rrbracket^g\}$$

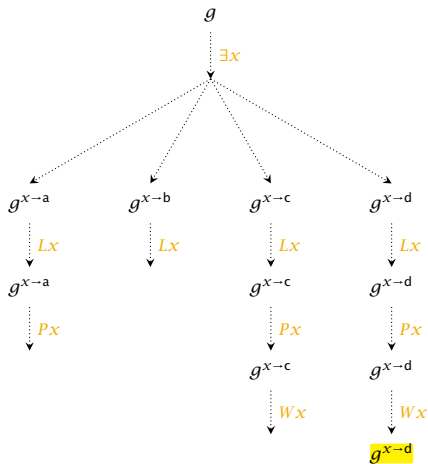
$$\blacktriangleright c[\exists x] := \{g^{x \rightarrow d} \mid g \in c, d \in D\}$$

$$\blacktriangleright c[\phi \wedge \psi] := c[\phi][\psi]$$

$$\blacktriangleright c[\neg\phi] := \{g \in c \mid \{g\}[\phi] = \emptyset\}$$

(Heim 1982, Groenendijk & Stokhof 1991a,b, Muskens 1996)

Cross-sentential anaphora: $(\exists x \wedge Lx \wedge Px) \wedge Wx$



Donkey anaphora

Implication is defined in the usual material way:

$$\blacktriangleright \phi \Rightarrow \psi := \neg(\phi \wedge \neg\psi)$$

Donkey anaphora follows. We get a “strong” reading, requiring every F to be G :

$$\begin{aligned}\blacktriangleright \exists xFx \Rightarrow Gx &= \neg(\exists xFx \wedge \neg Gx) \\ &= \neg\exists x(Fx \wedge \neg Gx)\end{aligned}$$

Effectful dynamic semantics

Dynamic semantics can be seen as a marriage of three **effects**:

- ▶ **Input**: reading from the context $c \mapsto \{g \in c \mid \llbracket F(x_1, \dots, x_n) \rrbracket^g\}$
- ▶ **Output**: writing to the context $c \mapsto \{g^{x \rightarrow d} \mid g \in c, d \in D\}$
- ▶ **Nondeterminism**: branching the context $c \mapsto \{g^{x \rightarrow d} \mid g \in c, d \in D\}$

Quantification and scope ambiguity imply one more effect:

- ▶ **Scope**: quantifiers can be infix into their semantic arguments

Effectful composition allows you to **decompose dynamic semantics** — that is, treat these effects as distinct and independent lexical and compositional primitives

Decomposing dynamic semantics via **adjunctions**

Nondeterministic state

Dynamic semantics as nondeterministic state: reading, writing, nondeterminism

$Da ::=$

$\eta x :=$

$\mu M :=$

Nondeterministic state

Dynamic semantics as nondeterministic state: reading, writing, nondeterminism

$$Da ::= s \rightarrow \{a \times s\}$$

$$\eta x :=$$

$$\mu M :=$$

Nondeterministic state

Dynamic semantics as nondeterministic state: reading, writing, nondeterminism

$$Da ::= s \rightarrow \{a \times s\}$$

$$\eta x := \lambda s. \{ \langle x, s \rangle \}$$

$$\mu M :=$$

Nondeterministic state

Dynamic semantics as nondeterministic state: reading, writing, nondeterminism

$$D a ::= s \rightarrow \{a \times s\}$$

$$\eta x := \lambda s. \{\langle x, s \rangle\}$$

$$\mu M := \lambda s. \bigcup_{\langle m, s' \rangle \in M s} m s'$$

Not as simple as it could be

Input, Output, Nondeterminism. Anything that does one does all, even if trivially.

- ▶ **she₀** := $\lambda s. \{(s_0, s)\}$
- ▶ **mary⁺** := $\lambda s. \{(m, s ++ m)\}$
- ▶ **someone** := $\lambda s. \{(x, s) \mid x : e\}$

Another sort of generalization to the worst case

Reading and Writing

We've seen Functors that **R**-ead values in, and Functors that **W**-rite values out

$$R a = s \rightarrow a$$

$$W a = a \times s$$

You might think that with the capacity to both push and pull things from a context, we ought to be able to model all kinds of anaphora:

- ▶ Polly walked in the park. She whistled.
- ▶ A linguist walked in the park. She whistled.
- ▶ If a linguist walked in the park, she whistled.
- ▶ When I have to clean, I don't want to. When I have to cook, I don't either.

Binding without scope

We'll make some very simple (in fact, variable-free) assumptions about meanings:

$$\mathbf{mary}^+ := \underbrace{(m, m)}_{We} \qquad \mathbf{she} := \underbrace{\lambda x. x}_{Re}$$

Both meanings are functorial (note: this W is not applicative). So we know a sentence like *Mary's mom saw her* will give rise to two meanings via (\bullet) .

$$\overset{\overline{FF}}{\rightsquigarrow} \lambda x. (\mathbf{saw} x (\mathbf{mom} m), m) : RWt \qquad \overset{\overline{FF}}{\rightsquigarrow} (\lambda x. \mathbf{saw} x (\mathbf{mom} m), m) : WRt$$

The second of these is **extremely interesting**. If we simply apply ϵ to it, we end up with binding. **No c-command or scope required.**

$$\overset{\epsilon}{\rightsquigarrow} \mathbf{saw} m (\mathbf{mom} m) : t$$

Adjunctions

R and **W** are **adjoint functors** (in particular, $W \dashv R$):

$$\begin{aligned}La \rightarrow b &\simeq a \rightarrow Rb \\ Wa \rightarrow b &\simeq a \rightarrow Rb \\ (a \times s) \rightarrow b &\simeq a \rightarrow s \rightarrow b\end{aligned}$$

Equivalently, **R** and **W** support the following operations:

$$\begin{array}{ll}\eta : a \rightarrow RWa & \epsilon : WRa \rightarrow a \\ : a \rightarrow s \rightarrow (a \times s) & : ((s \rightarrow a) \times s) \rightarrow a \\ = & =\end{array}$$

We have actually seen η before. It's just the η operation of an Applicative Functor! But ϵ , and the way it transforms types — destructively — are new.

Adjunctions

R and **W** are **adjoint functors** (in particular, $W \dashv R$):

$$\begin{aligned}La \rightarrow b &\cong a \rightarrow Rb \\ Wa \rightarrow b &\cong a \rightarrow Rb \\ (a \times s) \rightarrow b &\cong a \rightarrow s \rightarrow b\end{aligned}$$

Equivalently, **R** and **W** support the following operations:

$$\begin{array}{ll}\eta : a \rightarrow RWa & \epsilon : WRa \rightarrow a \\ : a \rightarrow s \rightarrow (a \times s) & : ((s \rightarrow a) \times s) \rightarrow a \\ = \lambda x. \lambda s. (x, s) & = \lambda (f, s). f s\end{array}$$

We have actually seen η before. It's just the η operation of an Applicative Functor!
But ϵ , and the way it transforms types — destructively — are new.

Adjunctions in Haskell

```
class (Functor f, Functor g) => Adjunction f g where
  {-# MINIMAL (unit, counit) | (leftAdjunct, rightAdjunct) #-}
  unit   :: a -> g (f a)           -- eta
  counit :: f (g a) -> a           -- epsilon
  phi    :: (f a -> b) -> a -> g b -- curry (e.g.)
  psi    :: (a -> g b) -> f a -> b -- uncurry (e.g.)

  -- unit/counit and phi/psi are interdefinable; an instance
  -- of Adjoint need only declare one or the other pair:
  unit = phi id
  counit = psi id
  phi c = fmap c . unit
  psi k = counit . fmap k
```

One last extension to the interpreter

Given $* : a \rightarrow b \rightarrow c$, and given $L \dashv R$:

$$\frac{(\lambda r'.l * r') \bullet r : Fc}{l : a \quad r : Fb} \bar{F}^* \qquad \frac{(\lambda l'.l' * r) \bullet l : Fc}{l : Fa \quad r : b} \bar{F}^*$$
$$\frac{(*) \bullet l \otimes r : Fc}{l : Fa \quad r : Fb} A^* \qquad \frac{\text{[redacted]} : c}{l : La \quad r : Rb} C^*$$

We can build stacks of effect contexts, merge contexts when effects are applicative, and bust out of an effectful context when we have an adjunction.

```
> everyone's mom saw her
> mary left and she whistled
> someone2 left and she whistled
> eclo someone2 saw someone2 and she chased her
```

One last extension to the interpreter

Given $* : a \rightarrow b \rightarrow c$, and given $L \dashv R$:

$$\frac{(\lambda r'. l * r') \bullet r : Fc}{l : a \quad r : Fb} \bar{F}^* \qquad \frac{(\lambda l'. l' * r) \bullet l : Fc}{l : Fa \quad r : b} \bar{F}^*$$
$$\frac{(*) \bullet l \otimes r : Fc}{l : Fa \quad r : Fb} A^* \qquad \frac{\epsilon((\lambda l'. (\lambda r'. l' * r') \bullet r) \bullet l) : c}{l : La \quad r : Rb} C^*$$

We can build stacks of effect contexts, merge contexts when effects are applicative, and bust out of an effectful context when we have an adjunction.

```
> everyone's mom saw her
> mary left and she whistled
> someone2 left and she whistled
> eclo someone2 saw someone2 and she chased her
```

Extending combine with **adjunctions**

```
data Mode
  = FA | BA | PM          -- Base modes           >, <, &
  | MR Mode | ML Mode    -- Meta-modes for functors fmap
  | AP Mode              -- Meta-modes for applicatives <*>
  | JN Mode              -- Meta-modes for monads   join
```

```
combine :: Type -> Type -> [(Mode, Type)]
combine l r = basic l r ++
  -- ...
  [ (CU op, c) | Comp f a <- [l], Comp g b <- [r],
    , adjunction f g, (op, c) <- combine a b ]
```

Crossover

Here again is our new \mathbf{C}^* rule for type-driven interpretation with adjunctions:

$$\frac{\epsilon((\lambda l'.(\lambda r'.l' * r') \bullet r) \bullet l) : c}{l : G a \quad r : F b} \mathbf{C}^*$$

This requires the G (i.e., W) to come from the left, and the F (i.e., R) to come from the right. This means the binder must occur to the left of the bindee!

This attested restriction on pronominal binding is known as **crossover**.

Crossover in the parser

Indeed, the parser generates binding derivations for:

- ▶ Mary saw her mom
- ▶ Everyone saw her mom
- ▶ Mary left and she whistled
- ▶ Someone left and she whistled

But not for any of the following:

- ▶ Her mom saw Mary
- ▶ Her mom saw everyone
- ▶ She whistled and Mary left
- ▶ She whistled and someone left

Like many programming languages natural language is evaluated **left-to-right**

From adjunctions to monads

```
class (Functor f, Functor g) => Adjunction f g where
  unit   :: a -> g (f a)           -- eta
  counit :: f (g a) -> a           -- epsilon
  phi    :: (f a -> b) -> a -> g b -- curry (e.g.)
  psi    :: (a -> g b) -> f a -> b -- uncurry (e.g.)
```

$F \dashv G$ implies that GF is a monad! We may deduce GF 's \bullet , η , and μ from $F \dashv G$.

- ▶ $\bullet: (a \rightarrow b) \rightarrow GFa \rightarrow GFb$ follows from functoriality of G and F
- ▶ $\eta: a \rightarrow GFa$ is the unit of the adjunction
- ▶ $\mu: GF GFa \rightarrow GFa$ is given by $G(\varepsilon)$

For RW: $\varepsilon \bullet_R M = \lambda s. (\lambda (m, s'). m s') (M s)$

Wrapping up

Denotations via functors

Expression	Type	Denotation
no cat	$Ce ::= (e \rightarrow t) \rightarrow t$	$\lambda c. \neg \exists x. \mathbf{cat} x \wedge c x$
the cat	$Me ::= e + \perp$	x if $\mathbf{cat} = \{x\}$ else $\#$
Sassy, a cat	$We ::= e \times t$	$\langle \mathbf{s}, \mathbf{cats} \rangle$
she	$Re ::= i \rightarrow e$	$\lambda i. \pi i$
which cat	$Se ::= \{e\}$	$\{x \mid \mathbf{cat} x\}$
SASSY	$Fe ::= e \times \{e\}$	$\langle \mathbf{s}, \{x \mid x \in D_e\} \rangle$
as for Sassy	$Te ::= s \rightarrow (e \times s)$	$\lambda s. \langle \mathbf{s}, s ++ \mathbf{s} \rangle$
a cat	$De ::= s \rightarrow \{e \times s\}$	$\lambda s. \{ \langle x, s ++ x \rangle \mid \mathbf{cat} x \}$
...

Meditate on the hoops you'd need to jump through to develop a theory of grammar in the standard mold that could handle all these effects (and more).

A hierarchy of interfaces for effects

-- Functors: computational effects accumulate

```
class Functor f where
```

```
  fmap :: (a -> b) -> f a -> f b
```

-- Applicatives: computations can be merged

```
class Functor f => Applicative f where
```

```
  pure  :: a -> f a
```

```
  (<*>) :: f (a -> b) -> f a -> f b
```

-- Monads: higher-order computations can be smooshed

```
class Applicative f => Monad f where
```

```
  join :: f (f a) -> f a
```

-- Adjunctions: mutually resolving computations

```
class (Functor f, Functor g) => Adjunction f g where
```

```
  unit  :: a -> g (f a) -- eta
```

```
  counit :: f (g a) -> a -- epsilon
```

Monads in the interpreter

Monads arise in two ways in the interpreter. First, many of our applicatives are additionally monads. The interpreter knows how to smooch them.

In addition, **every adjunction induces a monad**. Here, $W \dashv R$ decomposes the **State monad**, aka RW . ϵ does our smooching: $RWRa \rightsquigarrow RWa!$

Effect-driven interpretation

Effectful language is a fact of life.

- ▶ Sentences are full of pieces going well beyond function and argument.
- ▶ Effectful meanings **coexist** and **interact**. A grammar must allow this to happen in a flexible and extensible way.

Functors are a useful model for effectful composition.

- ▶ They help us see what different effects have in common, and use a small set of ingredients to generate highly intricate data.
- ▶ Previous theories of effects were operationalized in the object language. The resulting theories are inconvenient to work with and conceptually suspect.
- ▶ An extended type-driven interpreter does better, allowing simple calculation and efficient implementation, and non-trivial effect **interaction**.

Parting words

A lightweight compositional interface that extends familiar compositional semantic theories with effects is within reach.

We can extend type-driven interpretation, simply, with functors, applicatives, monads, adjoints, and possibly other effectful constructs, as the need arises.

Sometimes, this just gives a simpler, elegant, mathematically sound compositional infrastructure. Sometimes, it reveals new analyses of recalcitrant phenomena.

We hope to have given you a sense of the power and elegance of this approach, some of the empirical payoffs, and ways in which it simplifies semantics.

- Geach, Peter. 1962. *Reference and Generality*. Ithaca, NY: Cornell University Press.
- Groenendijk, Jeroen & Martin Stokhof. 1991a. Dynamic predicate logic. *Linguistics and Philosophy* 14(1). 39–100. <https://doi.org/10.1007/BF00628304>.
- Groenendijk, Jeroen & Martin Stokhof. 1991b. Two theories of dynamic semantics. In Jan van Eijck (ed.), *Logics in AI: European workshop JELIA '90 Amsterdam, The Netherlands, September 10–14, 1990 proceedings*, 55–64. Berlin, Heidelberg: Springer Berlin Heidelberg. <https://doi.org/10.1007/BFb0018433>.
- Heim, Irene. 1982. *The semantics of definite and indefinite noun phrases*. University of Massachusetts, Amherst Ph.D. thesis. <https://semanticsarchive.net/Archive/Tk0ZmYyY/>.
- Karttunen, Lauri. 1976. Discourse referents. In James D. McCawley (ed.), *Syntax and Semantics, volume 7: Notes from the Linguistic Underground*, 363–385. New York: Academic Press.
- Lewis, David. 1975. Adverbs of quantification. In Edward L. Keenan (ed.), *Formal Semantics of Natural Language*, 3–15. Cambridge: Cambridge University Press. <https://doi.org/10.1017/cbo9780511897696.003>.
- Muskens, Reinhard. 1996. Combining Montague semantics and discourse representation. *Linguistics and Philosophy* 19(2). 143–186. <https://doi.org/10.1007/BF00635836>.