

## Effectful composition in natural language semantics

Dylan Bumford (UCLA)

Simon Charlow (Rutgers)

June 27, 2018

NASSLLI 2018

Carnegie Mellon

## Overview

Techniques for structuring functional programs help us build semantic theories.

This isn't surprising. Our jobs are remarkably similar: compositionally building meaningful things from meaningful pieces. Often with some twists...

- ▶ Notions of effectful composition are common to linguistic semantics and (functional) programming: (applicative) functors, (co)monads, etc.
- ▶ Taking this idea seriously reveals recurring structural patterns in linguistic meaning composition, suggests unified analyses in varied domains.

## Combining effects

Composition of effects is a longstanding issue in programming contexts.

- ▶ We'll explore how various kinds of effects can be composed, in varied ways. Different kinds of composition are useful for different kinds of things.
- ▶ Extended case study: monadic dynamic semantics (“composing” reading, writing, nondeterminism), and its interaction with continuations.

(Effectful) composition

## Syntax and semantics

```
data Term = Con Int | Term :+: Term | Term **: Term
```

```
exp1 = Con 1 :+: (Con 2 **: Con 3)  -- exp1 :: Term
```

```
exp2 = (Con 1 :+: Con 2) **: Con 3  -- exp2 :: Term
```

```
eval (Con x)    = x
```

```
eval (a :+: b) = (eval a) + (eval b)
```

```
eval (a **: b) = (eval a) * (eval b)
```

```
-- eval exp1 = 7
```

```
-- eval exp2 = 9
```

## Types and (higher-order) functions

My interpreter says the following about the addition operation:

```
GHCi> :t (+)
```

```
(+) :: Int -> Int -> Int
```

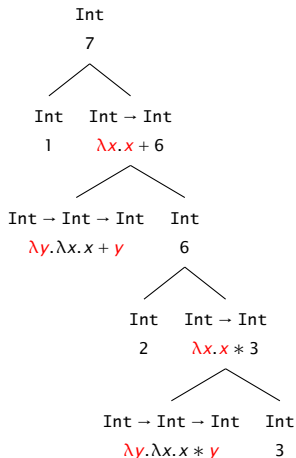
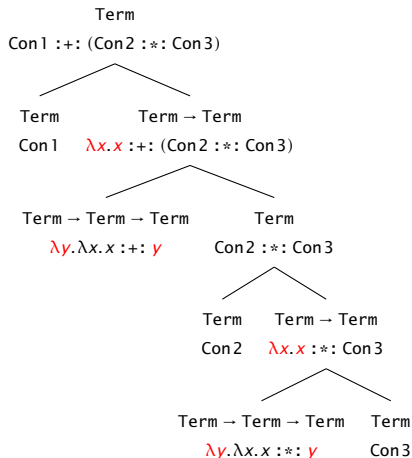
And it says the following about the corresponding term language operator:

```
GHCi> :t (:+::)
```

```
(::+) :: Term -> Term -> Term
```

Suggests another way to view term construction and evaluation.

## Construction and evaluation via iterated function application



## A baseline (extensional) semantic theory

Start with some basic types, and then ascend:

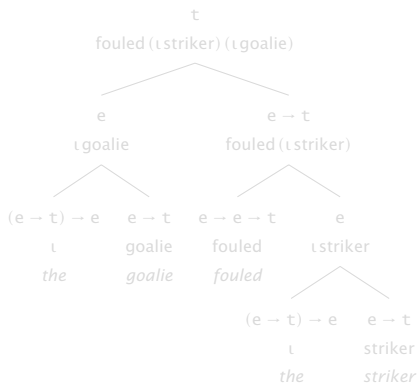
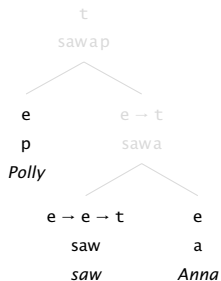
$$\tau ::= e \mid \mathbf{t} \mid \underbrace{\tau \rightarrow \tau}_{e \rightarrow \mathbf{t}, (e \rightarrow \mathbf{t}) \rightarrow \mathbf{t}, \dots}$$

Interpret binary combination via (type-driven) functional application:

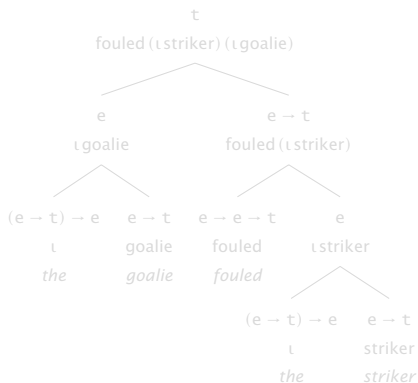
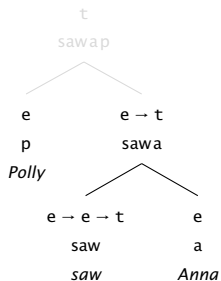
$$\llbracket \alpha \beta \rrbracket := \llbracket \alpha \rrbracket \llbracket \beta \rrbracket \text{ or } \llbracket \beta \rrbracket \llbracket \alpha \rrbracket, \text{ whichever is defined}$$



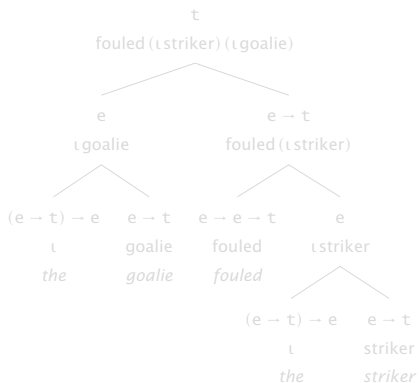
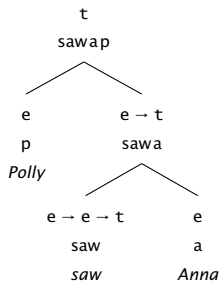
## A couple examples



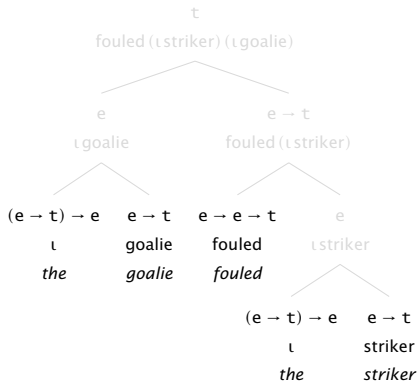
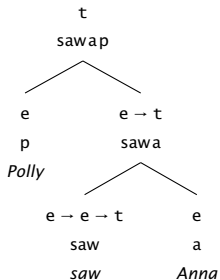
## A couple examples



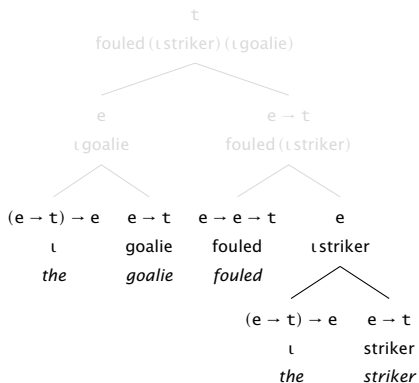
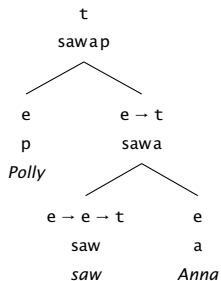
## A couple examples



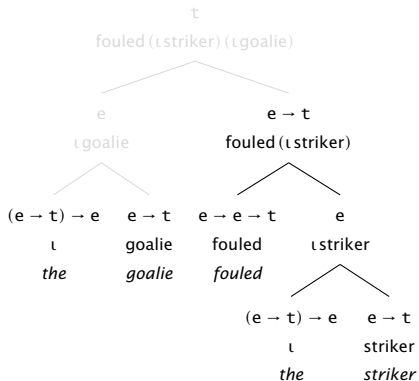
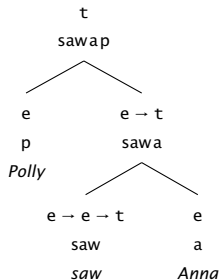
## A couple examples



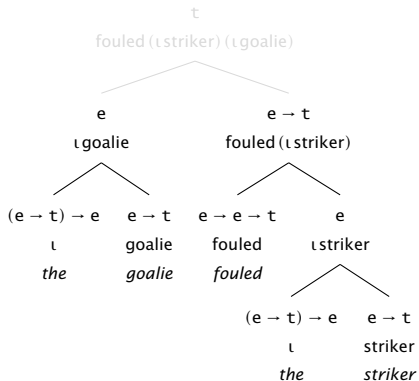
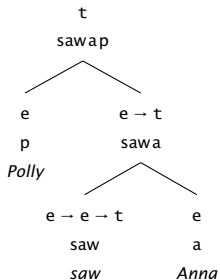
## A couple examples



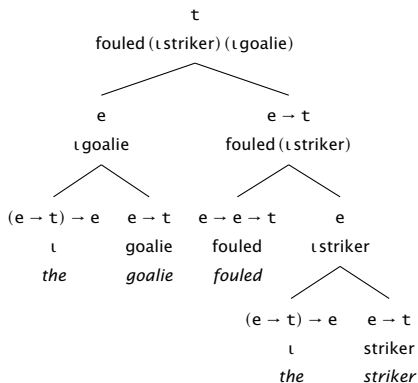
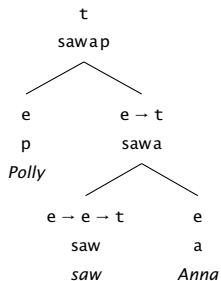
## A couple examples



## A couple examples



## A couple examples





## Assignment-dependence

Natural languages have free and bound pro-forms.

1. John saw her. I wouldn't \_ if I were you.
2. Everybody<sub>*i*</sub> did their<sub>*i*</sub> homework. When I'm supposed to work<sub>*i*</sub> I can't \_<sub>*i*</sub>.

Standardly: meanings depend on assignments (ways of valuing free variables).

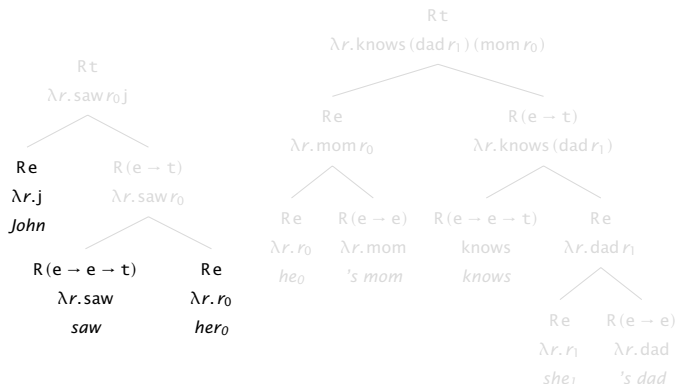
$$\sigma ::= e \mid \mathbf{t} \mid \sigma \rightarrow \sigma \qquad \tau ::= R \sigma ::= r \rightarrow \sigma$$

Interpret binary combination via **assignment-sensitive** functional application.

$$[[\alpha \beta]] := \lambda r. \underbrace{[[\alpha]]}_{R(b \rightarrow c)} r \underbrace{([[ \beta ]])}_{Rb} r$$

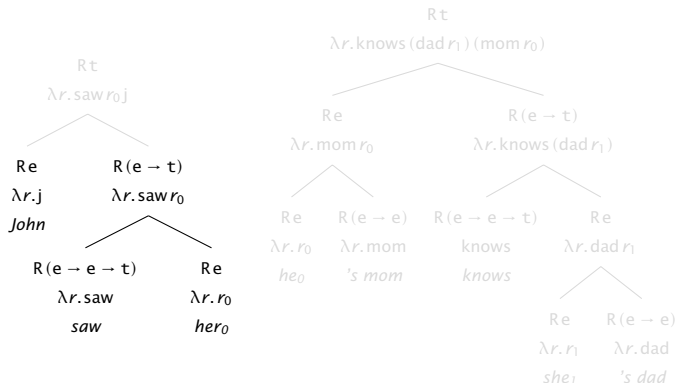
$\underbrace{\hspace{10em}}_{Rc}$

## A couple more examples



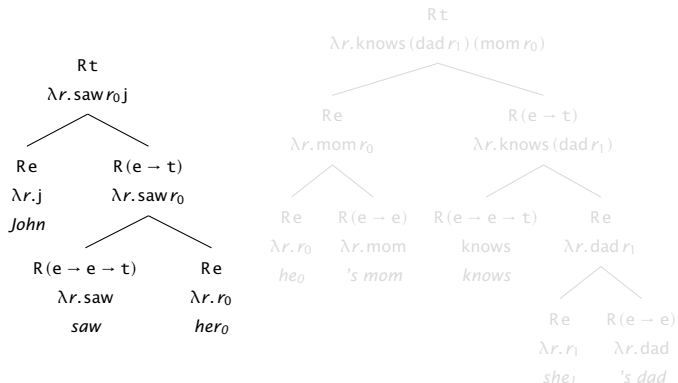
(Apply the result to a contextually furnished assignment to get a proposition.)

## A couple more examples



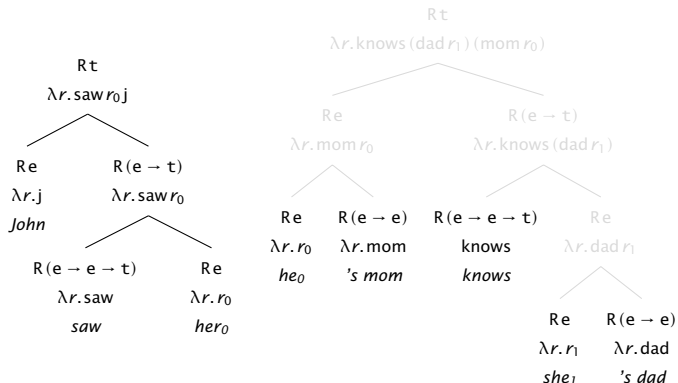
(Apply the result to a contextually furnished assignment to get a proposition.)

## A couple more examples



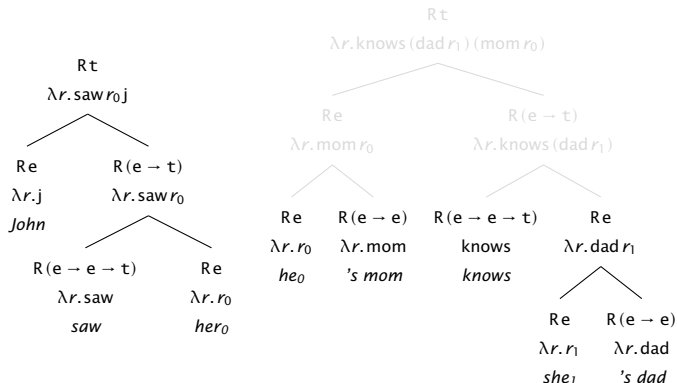
(Apply the result to a contextually furnished assignment to get a proposition.)

## A couple more examples



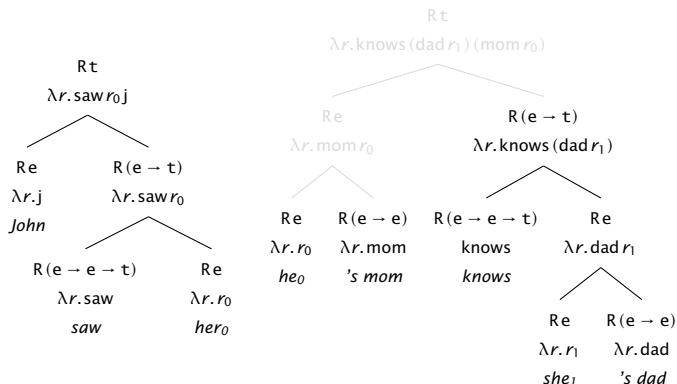
(Apply the result to a contextually furnished assignment to get a proposition.)

## A couple more examples



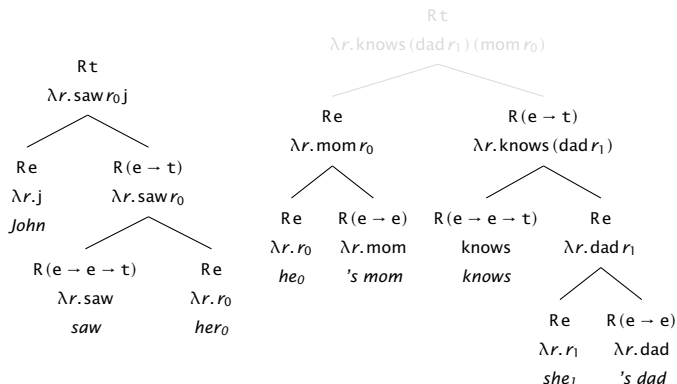
(Apply the result to a contextually furnished assignment to get a proposition.)

## A couple more examples



(Apply the result to a contextually furnished assignment to get a proposition.)

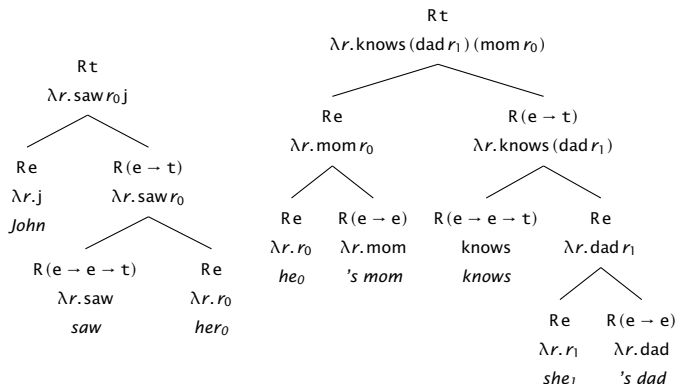
## A couple more examples



(Apply the result to a contextually furnished assignment to get a proposition.)



## A couple more examples



(Apply the result to a contextually furnished assignment to get a proposition.)

## Pulling out what matters

Key features of the standard approach to assignment-dependence:

- ▶ Uniformity: everything depends on an assignment (many things trivially).
- ▶ Enriched composition:  $[\cdot]$  stitches assignment-relative meanings together.

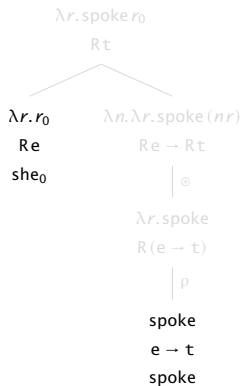
Here's another possibility: abstract out these key pieces, apply them on demand.

$$\underbrace{\rho x := \lambda r. x}_{\text{cf. } [\text{John}] := \lambda r. j}$$

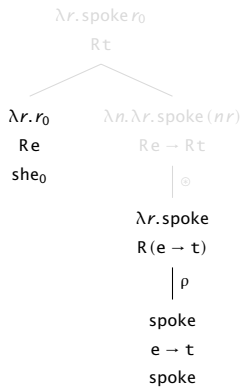
$$\underbrace{m \circledast n := \lambda r. m r (n r)}_{\text{cf. } [\alpha \beta] := \lambda r. [\alpha] r ([\beta] r)}$$

In terms of types,  $\rho :: a \rightarrow R a$ , and  $\circledast :: R(a \rightarrow b) \rightarrow R a \rightarrow R b$ .

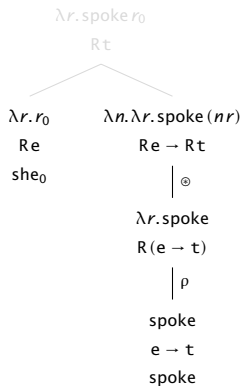
## An example



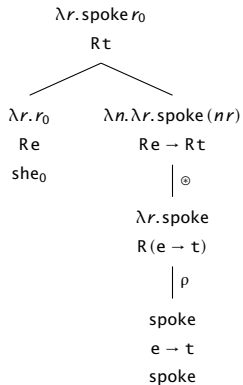
## An example



## An example



## An example



## Applicatives

R's  $\rho$  and  $\circledast$  make it an **applicative functor** (McBride & Paterson 2008, Kiselyov 2015). A type constructor  $F$  is applicative if it supports  $\rho$  and  $\circledast$  with these types...

$$\rho :: a \rightarrow Fa \qquad \circledast :: F(a \rightarrow b) \rightarrow Fa \rightarrow Fb$$

...Where  $\rho$  is a **trivial** way to inject something into the richer type characterized by  $F$ , and  $\circledast$  is function application **lifted** into  $F$ :

### Homomorphism

$$\rho f \circledast \rho x = \rho (f x)$$

### Identity

$$\rho (\lambda x. x) \circledast v = v$$

### Interchange

$$\rho (\lambda f. f x) \circledast u = u \circledast \rho x$$

### Composition

$$\rho (\circ) \circledast u \circledast v \circledast w = u \circledast (v \circledast w)$$

## Alternatives

It's common to treat question meanings as sets of possible answers:

3. Who ate the ham?  $\rightsquigarrow \{\text{ate } h \ x \mid x \in \text{human}\} :: S \ t$
4. Who ate what?  $\rightsquigarrow \{\text{ate } y \ x \mid x \in \text{human}, y \in \text{thing}\} :: S \ t$

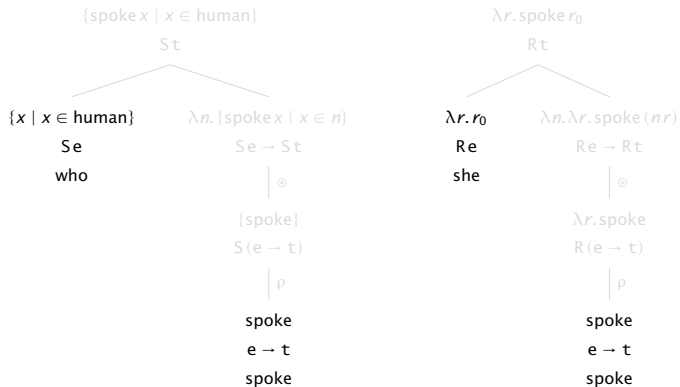
Naturally handled using another applicative functor, for sets (Hamblin 1973):

$$\underbrace{\rho x := \{x\}}_{\rho :: a \rightarrow S a} \quad \underbrace{m \otimes n := \{f x \mid f \in m, x \in n\}}_{\otimes :: S(a \rightarrow b) \rightarrow S a \rightarrow S b}$$

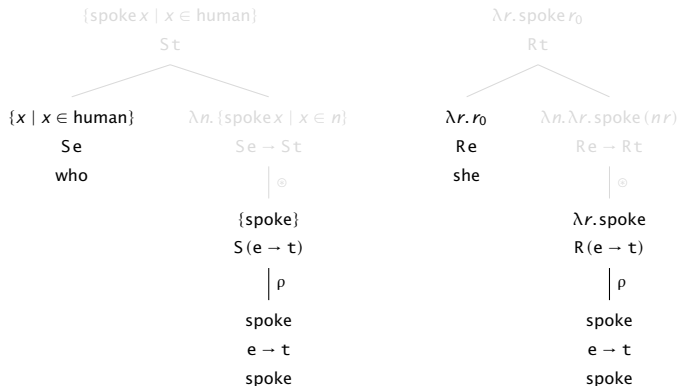
(See Charlow 2014, 2017 for more on this.)



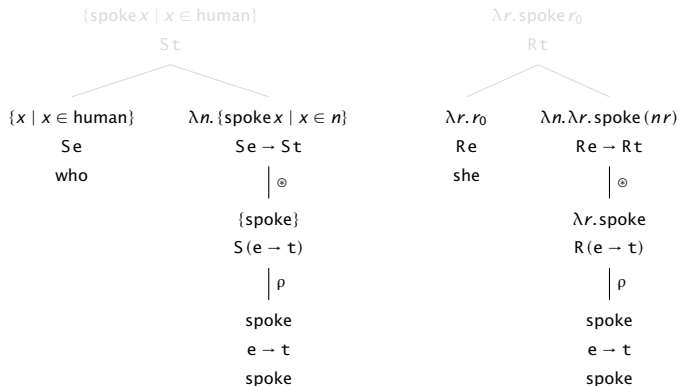
## Sample derivation, compared with context-sensitivity



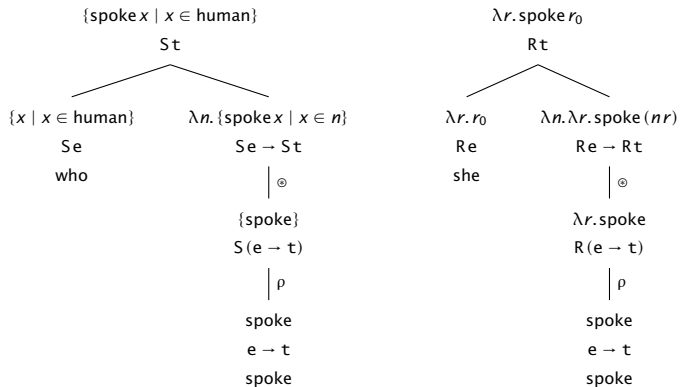
## Sample derivation, compared with context-sensitivity



## Sample derivation, compared with context-sensitivity



## Sample derivation, compared with context-sensitivity



## Supplementation<sup>1</sup>

Some expressions contribute information in a secondary “not-at-issue” register:

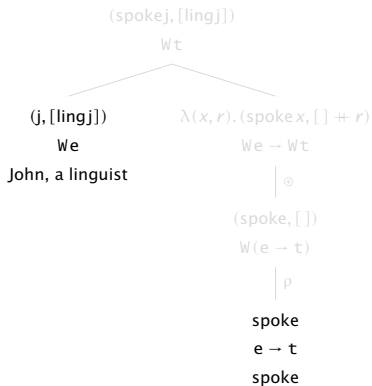
5. Joe, a linguist, lectured.  $\rightsquigarrow$  (lecturedj, [lingj]) :: Wt
6. Joe, a linguist, knows Mary, a philosopher.  $\rightsquigarrow$  (knows mj, [lingj, phil m]) :: Wt

Another example of an applicative functor, for supplements:

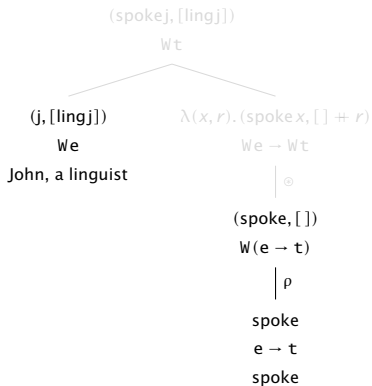
$$\underbrace{\rho x := (x, [])}_{\rho :: a \rightarrow Wa} \qquad \underbrace{(f, l) \circledast (x, r) := (fx, l \uparrow r)}_{\circledast :: W(a \rightarrow b) \rightarrow Wa \rightarrow Wb}$$

<sup>1</sup> Cf. Potts (2005), Giorgolo & Asudeh (2012).

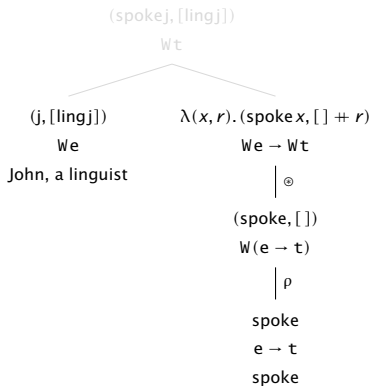
## Sample derivation: Supplementation



## Sample derivation: Supplementation

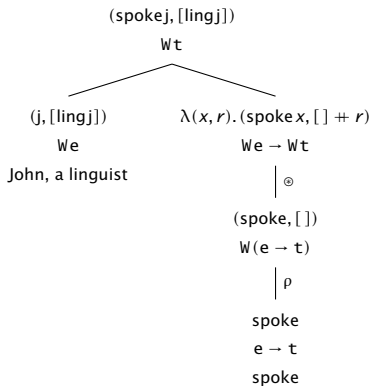


## Sample derivation: Supplementation





## Sample derivation: Supplementation



## Scope and continuations

Languages have quantificational expressions, and they take scope:

7. Every lecturer presented in a room on the third floor.

$\rightsquigarrow \forall(\lambda x. \exists(\lambda y. \text{pres } y x))$

$\rightsquigarrow \exists(\lambda y. \forall(\lambda x. \text{pres } y x))$

The relevant enrichment handles expressions with a scope (continuation):<sup>2</sup>

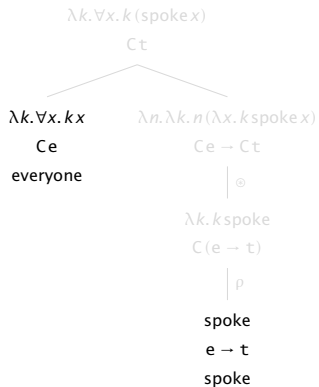
$$C_r a ::= (a \rightarrow r) \rightarrow r \quad \forall, \exists ::= C_t e = (e \rightarrow t) \rightarrow t$$

Yet another example of an applicative functor, for scope (continuations):

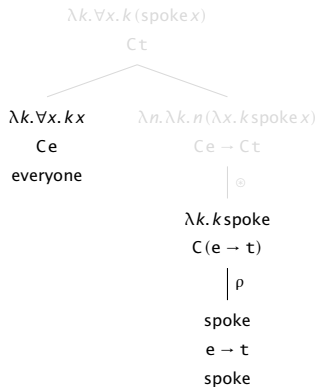
$$\rho x := \lambda k. k x \quad m \odot n := \lambda k. m(\lambda f. n(\lambda x. k(f x)))$$

<sup>2</sup>Barker (2002), Shan (2002), Shan & Barker (2006), Barker & Shan (2008, 2014), Charlow (2014).

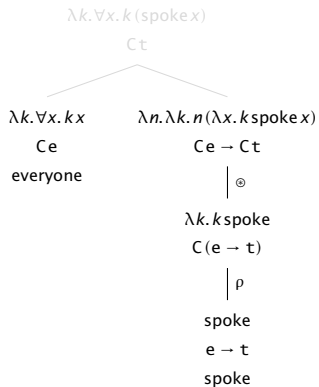
## Sample derivation: Scope



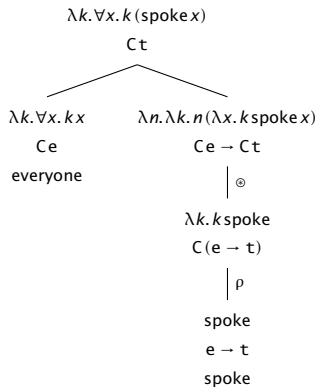
## Sample derivation: Scope



## Sample derivation: Scope



## Sample derivation: Scope



## Scope alternations via flexibility in $\odot$

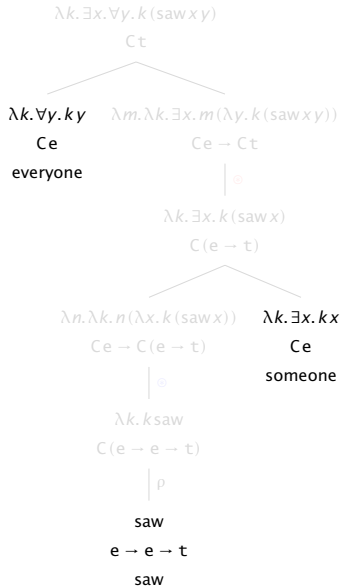
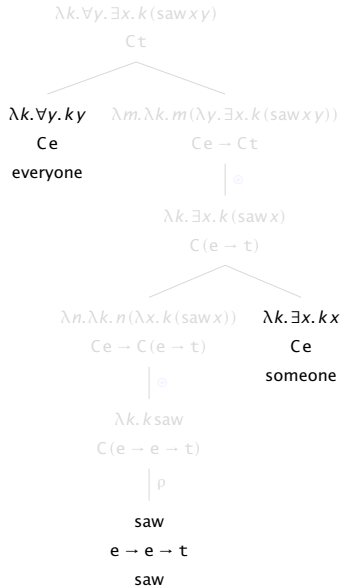
It turns out that the Continuations applicative is *non-commutative* in that it admits two  $\odot$ 's which evaluate their arguments in opposite orders.

$$\rho x := \lambda k. k x$$

$$\underbrace{m \odot n := \lambda k. m(\lambda f. n(\lambda x. k(f x)))}_{\text{function-first}}$$

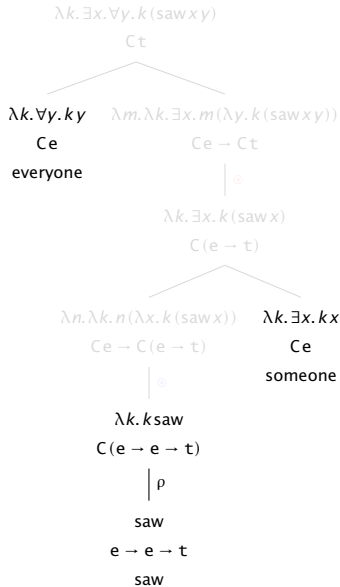
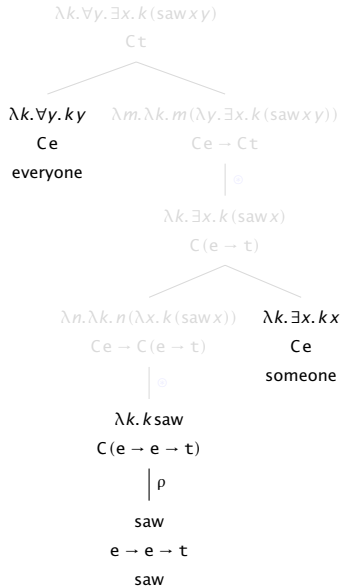
$$\underbrace{m \oplus n := \lambda k. n(\lambda x. m(\lambda f. k(f x)))}_{\text{argument-first}}$$

## A couple examples

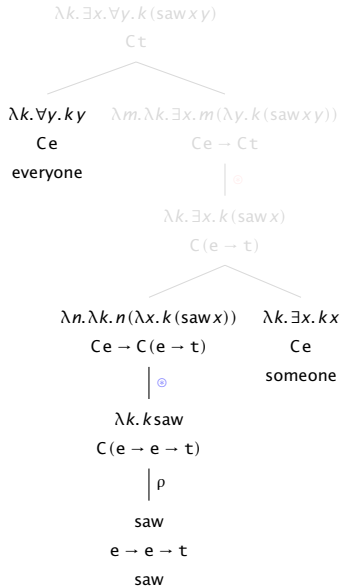
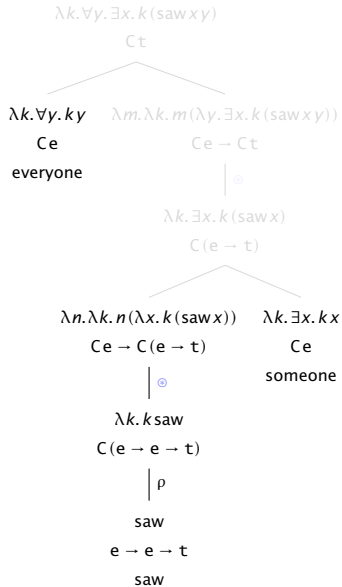




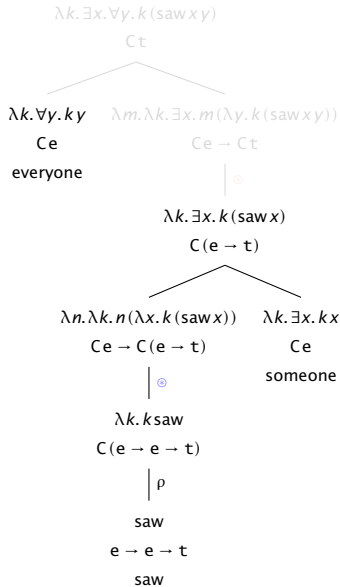
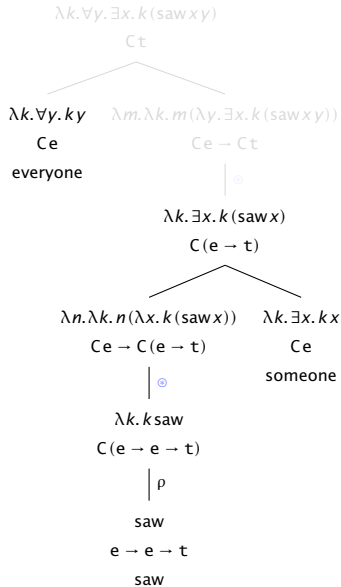
## A couple examples



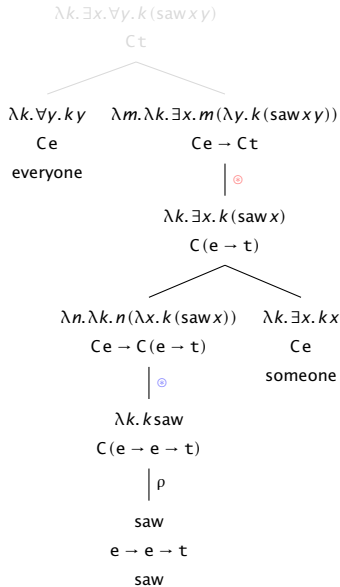
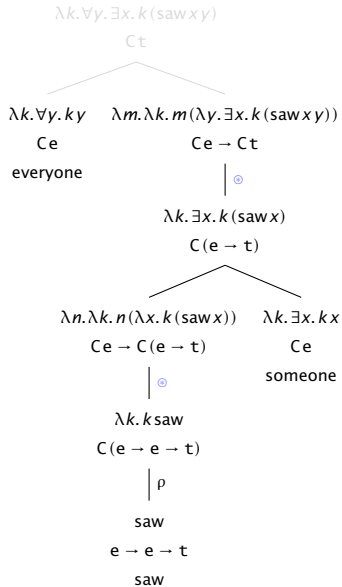
## A couple examples



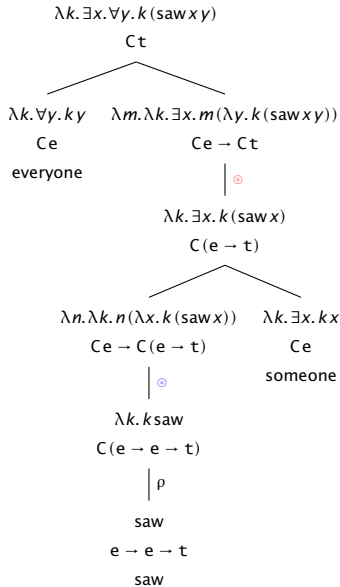
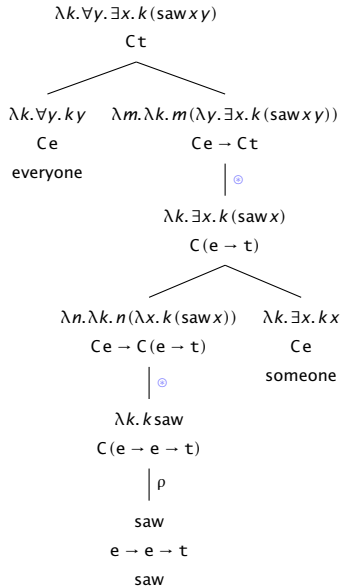
## A couple examples



## A couple examples



## A couple examples



## Corresponding notions in programs

- ▶ Reading: environment sensitivity, lexical scoping
- ▶ Writing: logging outputs, tracing the execution of a function
- ▶ Sets: denotational reification of nondeterminism
- ▶ Scope: control operators, aborting execution

## An applicative evaluator

```
class Functor f => Applicative f where
  pure  :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
```

```
eval :: Applicative f => Term -> f Int
```

```
eval (Con x) = pure x
```

```
eval (a :+: b) = pure (+) <*> (eval a) <*> (eval b)
```

```
eval (a ::: b) = pure (*) <*> (eval a) <*> (eval b)
```

Similar to enriching `[·]`. Another possibility, more closely related to the strategy we're using here, is having applicative combinators in the object language.

- ▶ (That's how Haskell programmers roll.)

## Reading and writing



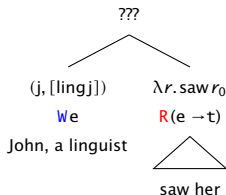
## Simultaneous effects

What do we have to do when we have to combine expressions that live in *different* effect regimes, given the modular nature of our  $\rho$  and  $\otimes$  combinators?

### 8. John, a linguist, saw her

John, a linguist :: **W**e

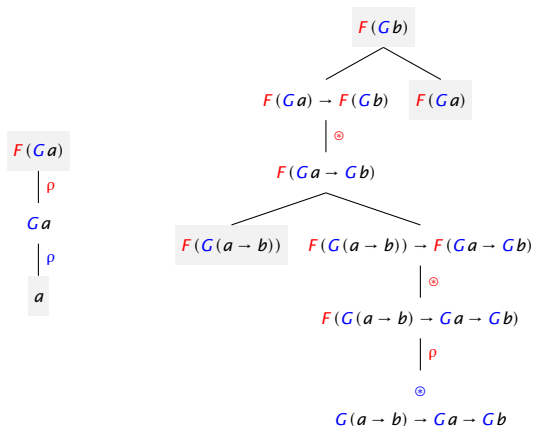
saw her :: **R**(e  $\rightarrow$  t)



*Let's not* invent new modes of combination for every combination of effects!

## Applicative functors compose

Fortunately, whenever you have two applicative functors, you're *guaranteed* to have two more, without adding anything to the grammar



## Composition with composition

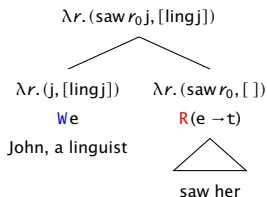
Here's what we get from taking the direct composition of **R** and **W**:

$$(\mathbf{R} \circ \mathbf{W}) a ::= r \rightarrow (a, [\mathbf{t}])$$

And here are the corresponding  $\rho$  and  $\otimes$  operations:

$$\rho x := \lambda r. (x, [ ])$$

$$m \otimes n := \lambda r. (f x, j \# k) \text{ where } (f, j) := m r \\ (x, k) := n r$$



## Some more composed applicatives<sup>3</sup>

Whenever  $F$  and  $G$  are applicative,  $F \circ G$  is too. Here, for  $R \circ S$ :

$$\begin{aligned}\rho x &:= \lambda r. \{x\} & m \odot n &:= \lambda r. \{f x \mid f \in m r, x \in n r\} \\ &= \rho(\rho x) & &= (\rho \odot) \odot m \odot n\end{aligned}$$

This corresponds to what is standardly called Alternative Semantics.

And here, for  $S \circ R$ :

$$\begin{aligned}\rho x &:= \{\lambda r. x\} & m \odot n &:= \{\lambda r. f r(x r) \mid f \in m, x \in n\} \\ &= \rho(\rho x) & &= (\rho \odot) \odot m \odot n\end{aligned}$$

<sup>3</sup>Cf. Rooth (1985), Kratzer & Shimoyama (2002), Romero & Novel (2013), Charlow (2017).

## Reading what's been written

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

9. Polly walked in the park. She whistled.

  
Write

  
Read

## Composing reading and writing actions

The reader/writer composition, with an entity-log:

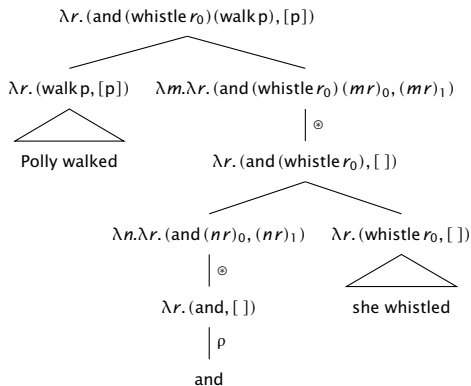
$$(R \circ W) a ::= r \rightarrow (a, [e])$$

And the corresponding  $\rho$  and  $\circledast$  operations again:

$$\rho x := \lambda r. (x, []) \quad m \circledast n := \lambda r. (f x, j \uparrow k) \textbf{ where } (f, j) := m r \\ (x, k) := n r$$

Not quite what we're after: the modified state output by  $m$  is not passed in to  $n$ .

## Failure to communicate



The pronoun Reads and the proper name Writes, but they don't coordinate.

## Another construction

But this nevertheless seems like the right structure to manage this sort of effect, and in fact, there is a *second* applicative for this type.

The State applicative:  $ST a ::= s \rightarrow (a, s)$

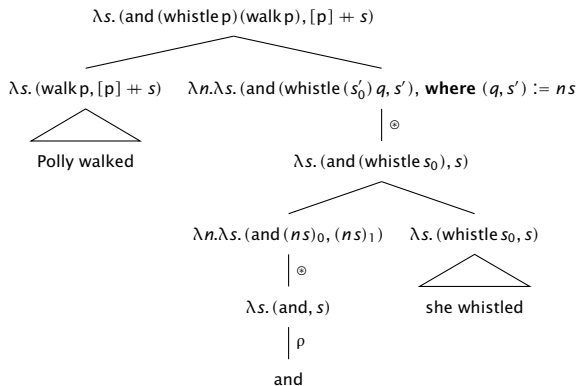
$$\rho x := \lambda s. (x, s) \quad m \circledast n := \lambda s. (f x, s'') \textbf{ where } (f, s') := m s \\ (x, s'') := n s'$$

$$\rho x := \lambda r. (x, [ ]) \quad m \circledast n := \lambda r. (f x, j \# k) \textbf{ where } (f, j) := m r \\ (x, k) := n r$$

Crucially, the modified state  $s'$  is passed into  $n$ .



## Successful communication



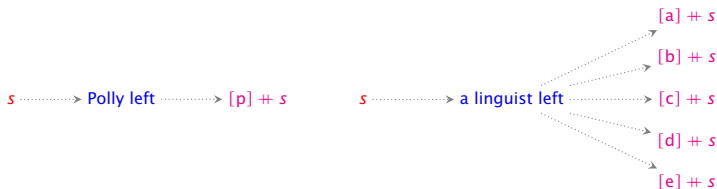
The proper name Writes something the pronoun Reads. Always nice.

## Indefinites<sup>4</sup>

True dynamic effects combine reading/writing with *nondeterminism*

10. Polly walked in the park. She whistled.

11. A linguist walked in the park. She whistled.



<sup>4</sup> Heim (1982), Barwise (1987), Rooth (1987), Groenendijk & Stokhof (1991), Muskens (1996), etc.

## Nondeterministic compositions with ST

Here are the two obvious options:  $S \circ ST$  and  $ST \circ S$

$ST \circ S$  :

$$\rho x := \{\lambda s. (x, s)\}$$

$$m \otimes n := \{\lambda s. (f x, s''), \text{ where } (f, s') := F s, (x, s'') := X s' \mid F \in m, X \in n\}$$

$S \circ ST$  :

$$\rho x := \lambda s. (\{x\}, s)$$

$$m \otimes n := \lambda s. (\{f x \mid f \in F, x \in X\}, s'') \text{ where, } (F, s') := m s \\ (X, s'') := n s'$$

## Problems with these compositions

However, independent of any issues with composition, neither of these types look like they're even up to the job

12. A linguist walked in the park. She whistled.

If a linguist  $:: s \rightarrow (\{e\}, s)$ , then we'll have to make a choice about which linguist ends up on the state

$$\llbracket \text{a linguist} \rrbracket \neq \lambda s. (\{x \mid \text{ling } x\}, [p] \uparrow s)$$

If a book she read  $:: \{s \rightarrow (a, s)\}$ , then we'll have to make a choice about how many books there are before we know who *she* refers to

$$\llbracket \text{a book she read} \rrbracket \neq \left\{ \lambda s. (x, [x] \uparrow s) \mid \text{book } x, \text{ read } \underline{\text{she}} x \right\}$$

## Nondeterministic state applicative

Fantastically, there is again *another* applicative hiding in these combinations of effects, but what we need is to interleave them!

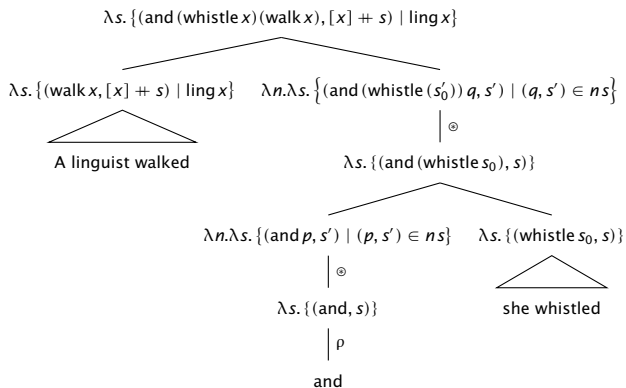
$$Da ::= s \rightarrow \{(a, s)\}$$

$$\rho x := \lambda s. \{(x, s)\} \quad m \otimes n := \lambda s. \{(f x, s'') \mid (f, s') \in m s, \\ (x, s'') \in n s'\}$$

$$\llbracket \text{a linguist} \rrbracket := \lambda s. \{(x, [x] \# s) \mid \text{ling } x\}$$

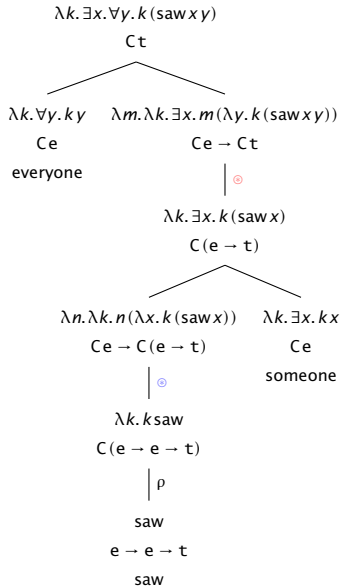
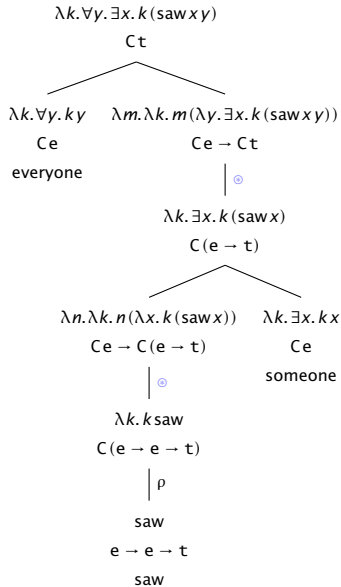
$$\llbracket \text{a book she read} \rrbracket := \lambda s. \{(x, [x] \# s) \mid \text{book } x, \text{ read } s_0 x\}$$

## Dynamics in action



## Scope and monads

## Scope interactions, refresher





## Too many indefinites

These derivations assumed 'a linguist' was a generalized quantifier: Ce

But in the dynamic section, we assumed 'a linguist' was a nondeterministic state modifier: De.

If this presentation is an advertisement for modularity, it would certainly be nice to hold onto this analysis of scope, even with 'a linguist' in a different type.

## Building toward a solution

First step: note that the continuation applicative works just as well for “static” GQs —  $C_t e = (e \rightarrow t) \rightarrow t$  — as it does for “dynamic” GQs —  $C_{Dt} e = (e \rightarrow Dt) \rightarrow Dt$

It's straightforward to define a meaning for universal quantifiers that has this shape:

$$\text{evOne} :: C_{Dt} e = (e \rightarrow Dt) \rightarrow Dt$$

But how are indefinites, type  $De$ , supposed to scopally interact with it?

Flipping  $\odot :: D(e \rightarrow t) \rightarrow De \rightarrow Dt$  and applying it to  $\text{aLing} :: De$  gives:

$$(\odot \text{aLing}) :: D(e \rightarrow t) \rightarrow Dt$$

## So close

This is so close to a dynamic GQ! If only  $\odot$  had the following type, we'd be golden:

$$(e \rightarrow Dt) \rightarrow De \rightarrow Dt$$

Actual type, as a reminder:

$$D(e \rightarrow t) \rightarrow De \rightarrow Dt$$

Many applicatives do in fact support a function of this type. Many do not. The ones that do are known as **monads**, and this function is given a special name:

$$\gg\! = :: Ma \rightarrow (a \rightarrow Mb) \rightarrow Mb$$

## Categorically

For those following along yesterday, any type with a  $\rho$  and  $\gg=$  also has well-behaved functions with these types

$$\odot :: F a \rightarrow (a \rightarrow b) \rightarrow F b \quad \mu :: F(F a) \rightarrow F a$$

in view of the fact that  $\mu(f \odot m) = m \gg= f$ .<sup>5</sup>

$\odot$  represents the functoriality of the  $F$ , and  $\mu$  is the monoid action taking  $F^2 \rightarrow F$

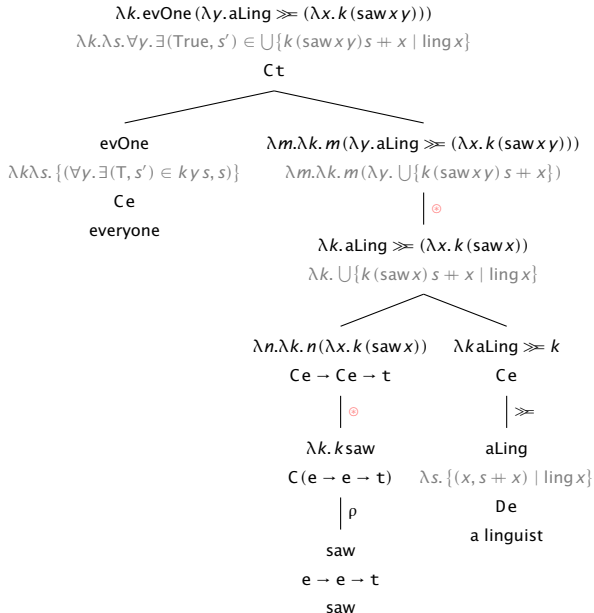
<sup>5</sup> Mere functors can, like monads, interact with continuations, but require the Indexed Continuations applicative. See Shan & Barker (2006), Barker & Shan (2014).

## The nondeterministic state monad

What does this  $\gg=$  function look like for our D?

$$m \gg= k := \lambda s. \bigcup \{k x s' \mid (x, s') \in m s\}$$

# Inverse scope derivation supported by $\gg$



## Scope ambiguity at the end of the day

Statically:

$$\forall (\lambda x. \exists (\lambda y. \text{saw } y \ x))$$
$$\exists (\lambda y. \forall (\lambda x. \text{saw } y \ x))$$

Dynamically:

$$\text{evOne} (\lambda x. \text{aLing} \gg \lambda y. \eta (\text{saw } y \ x))$$
$$\text{aLing} \gg \lambda y. \text{evOne} (\lambda x. \eta (\text{saw } y \ x))$$

## Semantic primitives?

State can be *decomposed* into reading and writing actions (cf. Shan 2001):

$$\text{Read } a ::= R \rightarrow a \qquad \text{Write } a ::= (a, R)$$

Read (aka  $R$ ) and Write are **adjoint functors** ( $\text{Write} \dashv \text{Read}$ ). In fact, Reading and Writing are adjoint *in virtue of the curry-uncurry isomorphisms*:

$$\begin{aligned} L a \rightarrow b &\simeq a \rightarrow R b \\ \text{Write } a \rightarrow b &\simeq a \rightarrow \text{Read } b \\ (a, R) \rightarrow b &\simeq a \rightarrow R \rightarrow b \end{aligned}$$

$L \dashv R$  iff  $RL$  is a monad (and  $LR$  a ‘comonad’)! What’s more,  $RL$  can compositionally *transform* any monad  $M$  into a ‘super-monad’  $RML$  with the functionality of  $R$  (e.g., reading),  $L$  (e.g., writing), and  $M$  (e.g., non-determinism).<sup>6</sup>

<sup>6</sup>This  $RL$  is the State monad, and  $R[ ]L$  is the State transformer (Liang, Hudak & Jones 1995) —  $RSL$  is none other than our  $D$ .  $LR$  is the Store comonad, useful for *structured meanings* (Krifka 1991, 2006).



- Barker, Chris. 2002. Continuations and the nature of quantification. *Natural Language Semantics* 10(3). 211-242. <https://doi.org/10.1023/A:1022183511876>.
- Barker, Chris & Chung-chieh Shan. 2008. Donkey anaphora is in-scope binding. *Semantics and Pragmatics* 1(1). 1-46. <https://doi.org/10.3765/sp.1.1>.
- Barker, Chris & Chung-chieh Shan. 2014. *Continuations and natural language*. Oxford: Oxford University Press. <https://doi.org/10.1093/acprof:oso/9780199575015.001.0001>.
- Barwise, Jon. 1987. Noun phrases, generalized quantifiers, and anaphora. In Peter Gärdenfors (ed.), *Generalized Quantifiers*, 1-29. Dordrecht: Reidel. [https://doi.org/10.1007/978-94-009-3381-1\\_1](https://doi.org/10.1007/978-94-009-3381-1_1).
- Charlow, Simon. 2014. *On the semantics of exceptional scope*. New York University Ph.D. thesis. <http://semanticsarchive.net/Archive/2JmMWRjY/>.
- Charlow, Simon. 2017. The scope of alternatives: Indefiniteness and islands. To appear in *Linguistics and Philosophy*. <http://ling.auf.net/lingbuzz/003302>.
- Giorgolo, Gianluca & Ash Asudeh. 2012. ( $M, \eta, \star$ ): Monads for conventional implicatures. In Ana Aguilar Guevara, Anna Chernilovskaya & Rick Nouwen (eds.), *Proceedings of Sinn und Bedeutung 16*, 265-278. MIT Working Papers in Linguistics. <http://mitwp1.mit.edu/open/sub16/Giorgolo.pdf>.
- Groenendijk, Jeroen & Martin Stokhof. 1991. Dynamic predicate logic. *Linguistics and Philosophy* 14(1). 39-100. <https://doi.org/10.1007/BF00628304>.

- Hamblin, C. L. 1973. Questions in Montague English. *Foundations of Language* 10(1). 41–53.
- Heim, Irene. 1982. *The semantics of definite and indefinite noun phrases*. University of Massachusetts, Amherst Ph.D. thesis. <http://semanticsarchive.net/Archive/Tk0ZmYyY/>.
- Kiselyov, Oleg. 2015. Applicative abstract categorial grammars. In Makoto Kanazawa, Lawrence S. Moss & Valeria de Paiva (eds.), *NLCS'15. Third workshop on natural language and computer science*, vol. 32 (EPIc Series), 29–38.
- Kratzer, Angelika & Junko Shimoyama. 2002. Indeterminate pronouns: The view from Japanese. In Yukio Otsu (ed.), *Proceedings of the Third Tokyo Conference on Psycholinguistics*, 1–25. Tokyo: Hituzi Syobo.
- Krifka, Manfred. 1991. A compositional semantics for multiple focus constructions. In Steve Moore & Adam Wyner (eds.), *Proceedings of Semantics and Linguistic Theory 1*, 127–158. Ithaca, NY: Cornell University.
- Krifka, Manfred. 2006. Association with focus phrases. In Valéria Molnár & Susanne Winkler (eds.), *The Architecture of Focus*, 105–136. Mouton de Gruyter.
- 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.
- McBride, Conor & Ross Paterson. 2008. Applicative programming with effects. *Journal of Functional Programming* 18(1). 1–13. <https://doi.org/10.1017/S0956796807006326>.
- Muskens, Reinhard. 1996. Combining Montague semantics and discourse representation. *Linguistics and Philosophy* 19(2). 143–186. <https://doi.org/10.1007/BF00635836>.

- Potts, Christopher. 2005. *The logic of conventional implicatures*. Oxford: Oxford University Press.  
<https://doi.org/10.1093/acprof:oso/9780199273829.001.0001>.
- 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.  
[https://doi.org/10.1057/9781137317247\\_7](https://doi.org/10.1057/9781137317247_7).
- Rooth, Mats. 1985. *Association with focus*. University of Massachusetts, Amherst Ph.D. thesis.
- Rooth, Mats. 1987. Noun phrase interpretation in Montague grammar, File Change Semantics, and situation semantics. In Peter Gärdenfors (ed.), *Generalized Quantifiers*, 237–269. Dordrecht: Reidel.  
[https://doi.org/10.1007/978-94-009-3381-1\\_9](https://doi.org/10.1007/978-94-009-3381-1_9).
- Shan, Chung-chieh. 2001. A variable-free dynamic semantics. In Robert van Rooij & Martin Stokhof (eds.), *Proceedings of the Thirteenth Amsterdam Colloquium*. University of Amsterdam.
- Shan, Chung-chieh. 2002. Monads for natural language semantics. In Kristina Striegnitz (ed.), *Proceedings of the ESSLLI 2001 Student Session*, 285–298. <http://arxiv.org/abs/cs/0205026>.
- Shan, Chung-chieh & Chris Barker. 2006. Explaining crossover and superiority as left-to-right evaluation. *Linguistics and Philosophy* 29(1). 91–134. <https://doi.org/10.1007/s10988-005-6580-7>.