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
```

Operations as higher-order functions

My interpreter says the following about the addition operation:

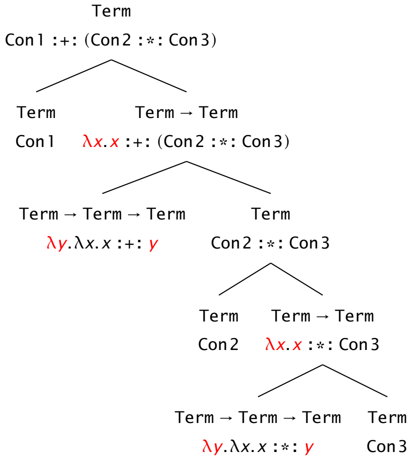
```
GHCi> :t (+)
(+) :: Int -> Int -> Int
```

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



Term
Con 1 :+: (Con 2 :∗: Con 3)

Term
Con 1

Term → Term
$\lambda x.x$:+: (Con 2 :∗: Con 3)

Term → Term → Term
$\lambda y.\lambda x.x$:+: y

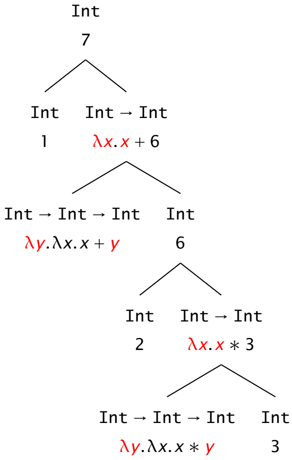
Term
Con 2 :∗: Con 3

Term
Con 2

Term → Term
$\lambda x.x$:∗: Con 3

Term → Term → Term
$\lambda y.\lambda x.x$:∗: y

Term
Con 3



Int
7

Int
1

Int → Int
$\lambda x.x + 6$

Int → Int → Int
$\lambda y.\lambda x.x + y$

Int
6

Int
2

Int → Int
$\lambda x.x * 3$

Int → Int → Int
$\lambda y.\lambda x.x * y$

Int
3

A baseline (extensional) semantic theory

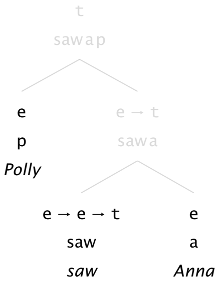
Start with some basic types, and then ascend:

$$\tau ::= e \mid t \mid \underbrace{\tau \to \tau}_{e\to t,\ (e\to t)\to t,\ \ldots}$$

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

$$[\![\alpha\ \beta]\!] := [\![\alpha]\!][\![\beta]\!] \text{ or } [\![\beta]\!][\![\alpha]\!], \text{ whichever is defined}$$

A couple examples

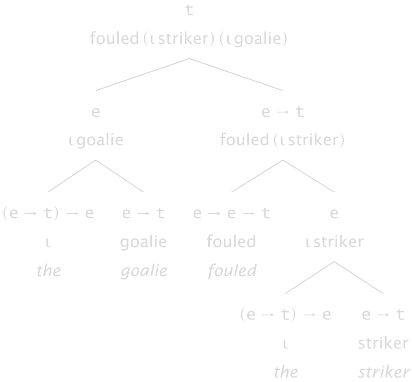

t
saw a p

e
p
Polly

e → t
saw a

e → e → t
saw
saw

e
a
Anna


t
fouled (ι striker) (ι goalie)

e
ι goalie

(e → t) → e
ι
the

e → t
goalie
goalie

e → t
fouled (ι striker)

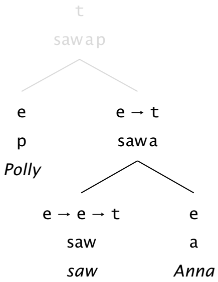
e → e → t
fouled
fouled

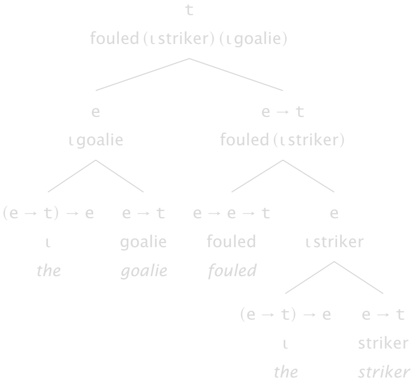
e
ι striker

(e → t) → e
ι
the

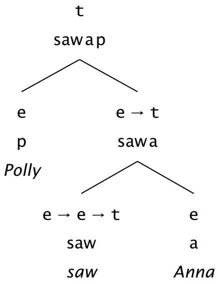
e → t
striker
striker

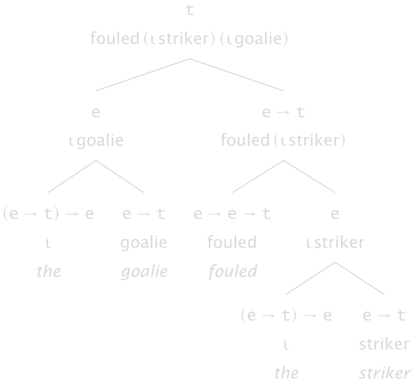
A couple examples



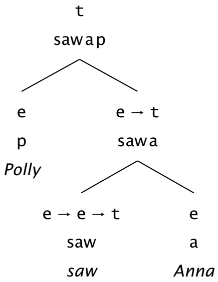


A couple examples





A couple examples



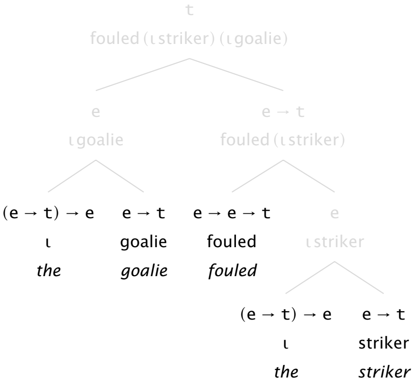
t
saw a p

e
p
Polly

e → t
saw a

e → e → t
saw
saw

e
a
Anna



t
fouled (ιstriker) (ιgoalie)

e
ιgoalie

(e → t) → e
ι
the

e → t
goalie
goalie

e → t
fouled (ιstriker)

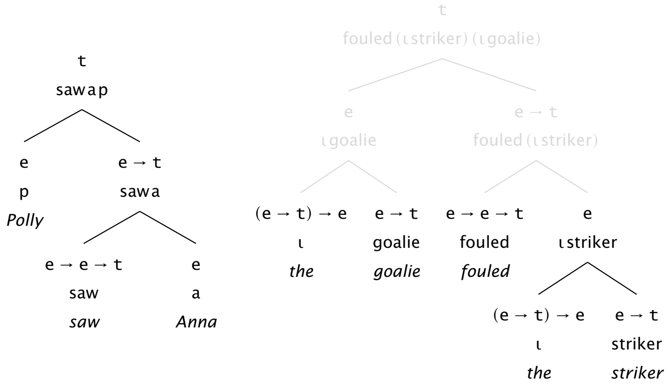
e → e → t
fouled
fouled

e
ιstriker

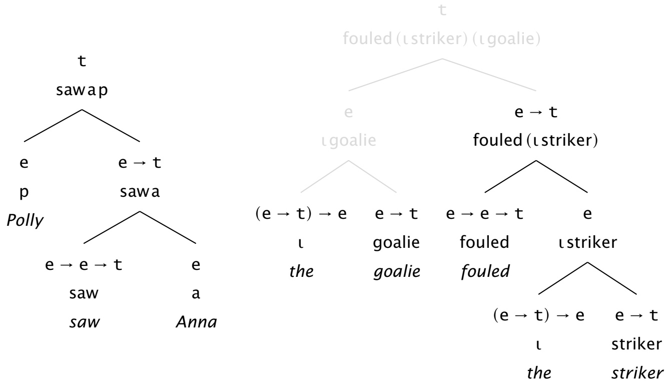
(e → t) → e
ι
the

e → t
striker
striker

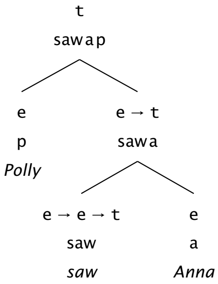
A couple examples



A couple examples



A couple examples



t

saw a p

e

p

Polly

e → t

saw a

e → e → t

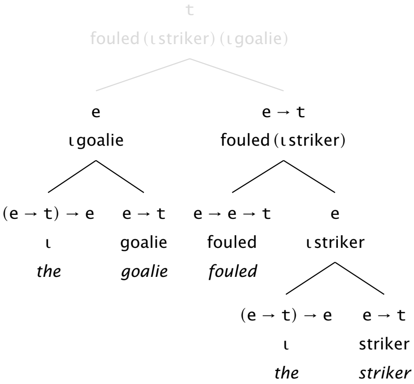
saw

saw

e

a

Anna



t

fouled (ιstriker) (ιgoalie)

e

ιgoalie

(e → t) → e

ι

the

e → t

goalie

goalie

e → t

fouled (ιstriker)

e → e → t

fouled

fouled

e

ιstriker

(e → t) → e

ι

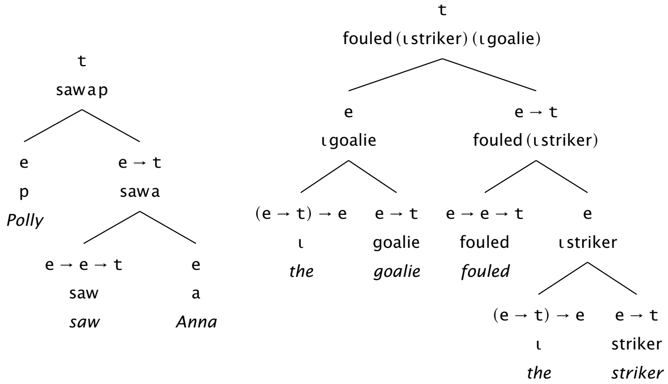
the

e → t

striker

striker

A couple examples



t
saw a p

e
p
Polly

e → t
saw a

e → e → t
saw
saw

e
a
Anna

t
fouled (ι striker) (ι goalie)

e
ι goalie

(e → t) → e
ι
the

e → t
goalie
goalie

e → t
fouled (ι striker)

e → e → t
fouled
fouled

e
ι striker

(e → t) → e
ι
the

e → t
striker
striker

Assignment-dependence

Natural languages have free and bound pro-forms.

1. John saw her. I wouldn't _ if I were you.
2. Everybody$_i$ did their$_i$ homework. When I'm supposed to work$_i$ I can't $\_{}_i$.

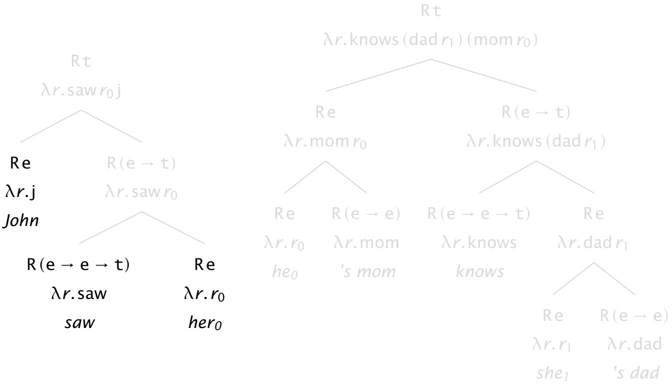
Standardly: meanings depend on assignments (ways of valuing free variables).

$$\sigma ::= \mathsf{e} \mid \mathsf{t} \mid \sigma \to \sigma \qquad\qquad \tau ::= \mathsf{R}\sigma ::= \mathsf{r} \to \sigma$$

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

$$[\![\alpha\ \beta]\!] := \underbrace{\lambda r.\, \underbrace{[\![\alpha]\!]}_{\mathsf{R}\,(b\to c)}\, r\, (\underbrace{[\![\beta]\!]}_{\mathsf{R}\,b}\, r)}_{\mathsf{R}\,c}$$

A couple more examples



R t
$\lambda r.\,\mathsf{saw}\, r_0\, \mathsf{j}$

R e
$\lambda r.\mathsf{j}$
John

R (e → t)
$\lambda r.\,\mathsf{saw}\, r_0$

R (e → e → t)
$\lambda r.\,\mathsf{saw}$
saw

R e
$\lambda r.\, r_0$
her$_0$

R t
$\lambda r.\,\mathsf{knows}\,(\mathsf{dad}\, r_1)\,(\mathsf{mom}\, r_0)$

R e
$\lambda r.\,\mathsf{mom}\, r_0$

R (e → t)
$\lambda r.\,\mathsf{knows}\,(\mathsf{dad}\, r_1)$

R e
$\lambda r.\, r_0$
he$_0$

R (e → e)
$\lambda r.\,\mathsf{mom}$
's mom

R (e → e → t)
$\lambda r.\,\mathsf{knows}$
knows

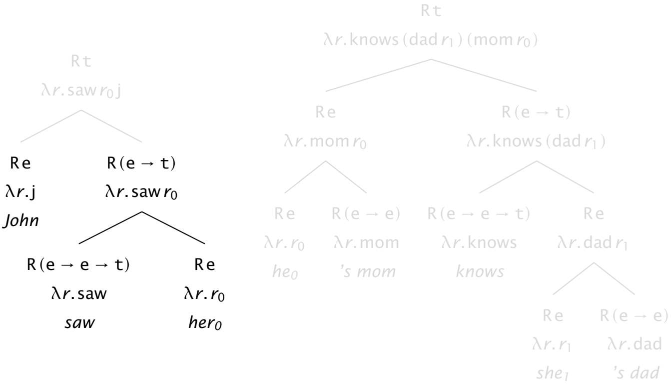
R e
$\lambda r.\,\mathsf{dad}\, r_1$

R e
$\lambda r.\, r_1$
she$_1$

R (e → e)
$\lambda r.\,\mathsf{dad}$
's dad

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

A couple more examples



R t

$\lambda r.\,\mathrm{saw}\; r_0\; \mathrm{j}$

R e

$\lambda r.\mathrm{j}$

John

R (e → t)

$\lambda r.\,\mathrm{saw}\; r_0$

R (e → e → t)

$\lambda r.\,\mathrm{saw}$

saw

R e

$\lambda r.\, r_0$

her$_0$

R t

$\lambda r.\,\mathrm{knows}\,(\mathrm{dad}\; r_1)\,(\mathrm{mom}\; r_0)$

R e

$\lambda r.\,\mathrm{mom}\; r_0$

R e

$\lambda r.\, r_0$

he$_0$

R (e → e)

$\lambda r.\,\mathrm{mom}$

's mom

R (e → t)

$\lambda r.\,\mathrm{knows}\,(\mathrm{dad}\; r_1)$

R (e → e → t)

$\lambda r.\,\mathrm{knows}$

knows

R e

$\lambda r.\,\mathrm{dad}\; r_1$

R e

$\lambda r.\, r_1$

she$_1$

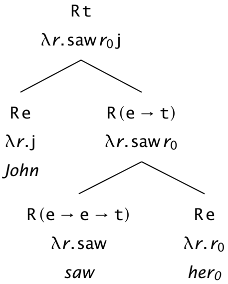
R (e → e)

$\lambda r.\,\mathrm{dad}$

's dad

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

A couple more examples



$\mathtt{R\,t}$

$\lambda r.\,\mathsf{saw}\,r_0\,\mathsf{j}$

$\mathtt{R\,e}$

$\lambda r.\,\mathsf{j}$

John

$\mathtt{R\,(e \to t)}$

$\lambda r.\,\mathsf{saw}\,r_0$

$\mathtt{R\,(e \to e \to t)}$

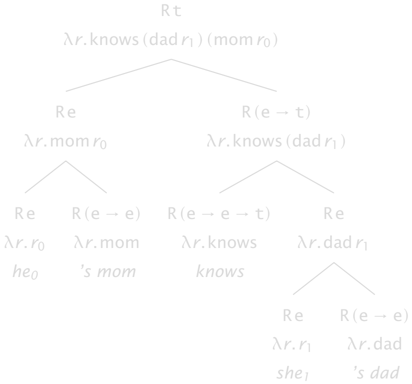
$\lambda r.\,\mathsf{saw}$

saw

$\mathtt{R\,e}$

$\lambda r.\,r_0$

her$_0$



$\mathtt{R\,t}$

$\lambda r.\,\mathsf{knows}\,(\mathsf{dad}\,r_1)\,(\mathsf{mom}\,r_0)$

$\mathtt{R\,e}$

$\lambda r.\,\mathsf{mom}\,r_0$

$\mathtt{R\,e}$

$\lambda r.\,r_0$

he$_0$

$\mathtt{R\,(e \to e)}$

$\lambda r.\,\mathsf{mom}$

's mom

$\mathtt{R\,(e \to t)}$

$\lambda r.\,\mathsf{knows}\,(\mathsf{dad}\,r_1)$

$\mathtt{R\,(e \to e \to t)}$

$\lambda r.\,\mathsf{knows}$

knows

$\mathtt{R\,e}$

$\lambda r.\,\mathsf{dad}\,r_1$

$\mathtt{R\,e}$

$\lambda r.\,r_1$

she$_1$

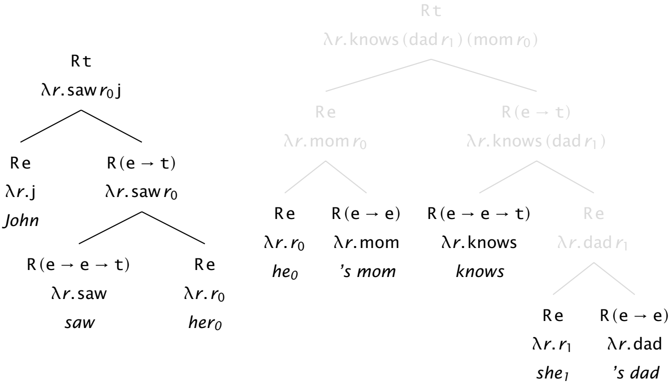
$\mathtt{R\,(e \to e)}$

$\lambda r.\,\mathsf{dad}$

's dad

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

A couple more examples



$\mathtt{R\,t}$
$\lambda r.\,\text{saw}\,r_0\,\text{j}$

$\mathtt{R\,e}$
$\lambda r.\text{j}$
John

$\mathtt{R\,(e \to t)}$
$\lambda r.\,\text{saw}\,r_0$

$\mathtt{R\,(e \to e \to t)}$
$\lambda r.\,\text{saw}$
saw

$\mathtt{R\,e}$
$\lambda r.\,r_0$
her$_0$

$\mathtt{R\,t}$
$\lambda r.\,\text{knows}\,(\text{dad}\,r_1)\,(\text{mom}\,r_0)$

$\mathtt{R\,e}$
$\lambda r.\,\text{mom}\,r_0$

$\mathtt{R\,(e \to t)}$
$\lambda r.\,\text{knows}\,(\text{dad}\,r_1)$

$\mathtt{R\,e}$
$\lambda r.\,r_0$
he$_0$

$\mathtt{R\,(e \to e)}$
$\lambda r.\,\text{mom}$
's mom

$\mathtt{R\,(e \to e \to t)}$
$\lambda r.\,\text{knows}$
knows

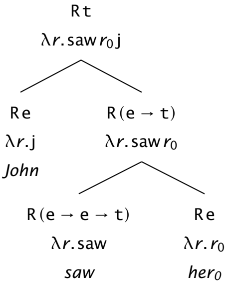
$\mathtt{R\,e}$
$\lambda r.\,\text{dad}\,r_1$

$\mathtt{R\,e}$
$\lambda r.\,r_1$
she$_1$

$\mathtt{R\,(e \to e)}$
$\lambda r.\,\text{dad}$
's dad

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

A couple more examples



R t

$\lambda r.\,\mathsf{saw}\, r_0\, \mathsf{j}$

R e

$\lambda r.\,\mathsf{j}$

John

R (e → t)

$\lambda r.\,\mathsf{saw}\, r_0$

R (e → e → t)

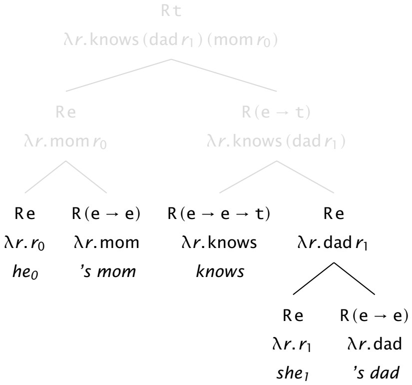
$\lambda r.\,\mathsf{saw}$

saw

R e

$\lambda r.\, r_0$

her$_0$



R t

$\lambda r.\,\mathsf{knows}\,(\mathsf{dad}\, r_1)\,(\mathsf{mom}\, r_0)$

R e

$\lambda r.\,\mathsf{mom}\, r_0$

R (e → t)

$\lambda r.\,\mathsf{knows}\,(\mathsf{dad}\, r_1)$

R e

$\lambda r.\, r_0$

he$_0$

R (e → e)

$\lambda r.\,\mathsf{mom}$

's mom

R (e → e → t)

$\lambda r.\,\mathsf{knows}$

knows

R e

$\lambda r.\,\mathsf{dad}\, r_1$

R e

$\lambda r.\, r_1$

she$_1$

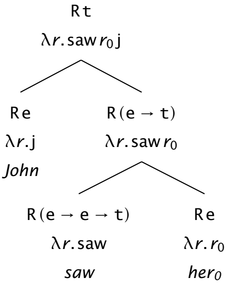
R (e → e)

$\lambda r.\,\mathsf{dad}$

's dad

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

A couple more examples



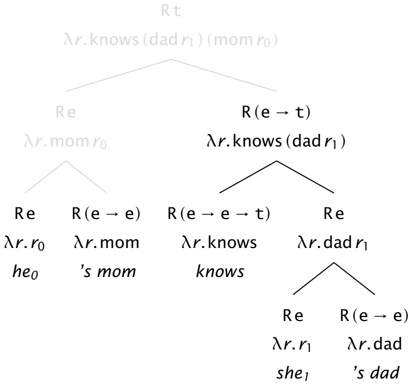
R t
$\lambda r.\,\mathrm{saw}\, r_0\, \mathrm{j}$

R e
$\lambda r.\mathrm{j}$
John

R (e → t)
$\lambda r.\,\mathrm{saw}\, r_0$

R (e → e → t)
$\lambda r.\,\mathrm{saw}$
saw

R e
$\lambda r.\, r_0$
her$_0$



R t
$\lambda r.\,\mathrm{knows}\,(\mathrm{dad}\, r_1)\,(\mathrm{mom}\, r_0)$

R e
$\lambda r.\,\mathrm{mom}\, r_0$

R e
$\lambda r.\, r_0$
he$_0$

R (e → e)
$\lambda r.\,\mathrm{mom}$
's mom

R (e → t)
$\lambda r.\,\mathrm{knows}\,(\mathrm{dad}\, r_1)$

R (e → e → t)
$\lambda r.\,\mathrm{knows}$
knows

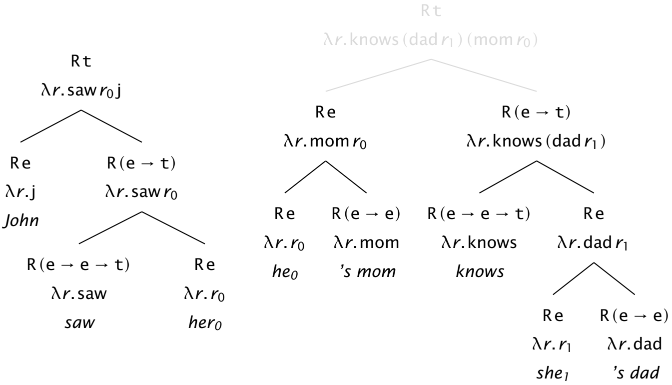
R e
$\lambda r.\,\mathrm{dad}\, r_1$

R e
$\lambda r.\, r_1$
she$_1$

R (e → e)
$\lambda r.\,\mathrm{dad}$
's dad

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

A couple more examples



R t

$\lambda r.\,\mathrm{saw}\, r_0\, \mathrm{j}$

R e

$\lambda r.\,\mathrm{j}$

John

R (e → t)

$\lambda r.\,\mathrm{saw}\, r_0$

R (e → e → t)

$\lambda r.\,\mathrm{saw}$

saw

R e

$\lambda r.\, r_0$

her$_0$

R t

$\lambda r.\,\mathrm{knows}\,(\mathrm{dad}\, r_1)\,(\mathrm{mom}\, r_0)$

R e

$\lambda r.\,\mathrm{mom}\, r_0$

R e

$\lambda r.\, r_0$

he$_0$

R (e → e)

$\lambda r.\,\mathrm{mom}$

's mom

R (e → t)

$\lambda r.\,\mathrm{knows}\,(\mathrm{dad}\, r_1)$

R (e → e → t)

$\lambda r.\,\mathrm{knows}$

knows

R e

$\lambda r.\,\mathrm{dad}\, r_1$

R e

$\lambda r.\, r_1$

she$_1$

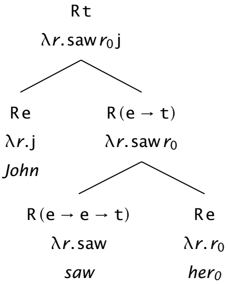
R (e → e)

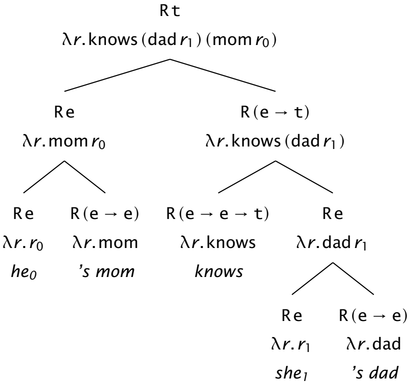
$\lambda r.\,\mathrm{dad}$

's dad

(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:

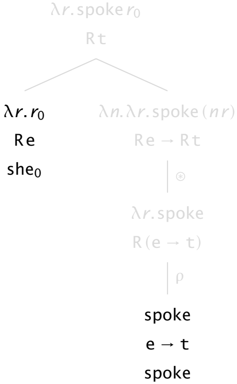
- Uniformity: everything depends on an assignment (many things trivially).
- Enriched composition: $[\![\cdot]\!]$ stitches assignment-relative meanings together.

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

$$\underbrace{\rho\, x := \lambda r.\, x}_{\text{cf. } [\![\text{John}]\!] := \lambda r.\text{j}} \qquad \underbrace{m \circledast n := \lambda r.\, m\, r\, (n\, r)}_{\text{cf. } [\![\alpha\ \beta]\!] := \lambda r.[\![\alpha]\!]\, r\, ([\![\beta]\!]\, r)}$$

In terms of types, $\rho :: a \to \mathrm{R}\, a$, and $\circledast :: \mathrm{R}\,(a \to b) \to \mathrm{R}\, a \to \mathrm{R}\, b$.

An example



$\lambda r.\,\mathsf{spoke}\,r_0$

$\mathsf{R}\,\mathsf{t}$

$\lambda r.\,r_0$

$\mathsf{R}\,\mathsf{e}$

she_0

$\lambda n.\lambda r.\,\mathsf{spoke}\,(n\,r)$

$\mathsf{R}\,\mathsf{e} \to \mathsf{R}\,\mathsf{t}$

$\circledast$

$\lambda r.\,\mathsf{spoke}$

$\mathsf{R}\,(\mathsf{e} \to \mathsf{t})$

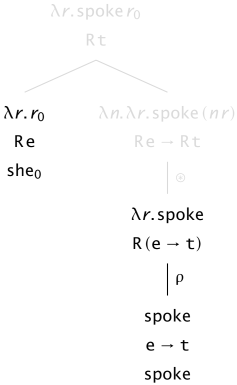
ρ

spoke

$\mathsf{e} \to \mathsf{t}$

spoke

An example



$\lambda r.\,\mathsf{spoke}\,r_0$

$\mathsf{R\,t}$

$\lambda r.\,r_0$

$\mathsf{R\,e}$

she_0

$\lambda n.\lambda r.\,\mathsf{spoke}\,(n\,r)$

$\mathsf{R\,e} \to \mathsf{R\,t}$

$\circledast$

$\lambda r.\,\mathsf{spoke}$

$\mathsf{R\,(e \to t)}$

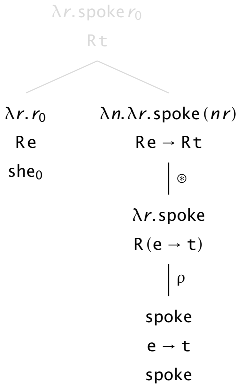
ρ

spoke

$\mathsf{e \to t}$

spoke

An example



$\lambda r.\,\mathsf{spoke}\,r_0$

$\mathtt{R\,t}$

$\lambda r.\,r_0$

$\mathtt{R\,e}$

she_0

$\lambda n.\lambda r.\,\mathsf{spoke}\,(n\,r)$

$\mathtt{R\,e} \to \mathtt{R\,t}$

$\circledast$

$\lambda r.\,\mathsf{spoke}$

$\mathtt{R}\,(\mathtt{e} \to \mathtt{t})$

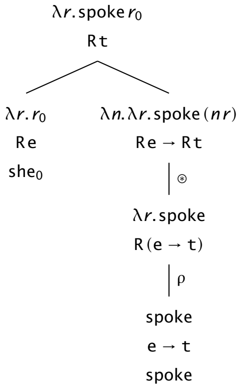
ρ

spoke

$\mathtt{e} \to \mathtt{t}$

spoke

An example



$\lambda r.\,\mathsf{spoke}\,r_0$

$\mathtt{R\,t}$

$\lambda r.\,r_0$

$\mathtt{R\,e}$

she_0

$\lambda n.\lambda r.\,\mathsf{spoke}\,(n\,r)$

$\mathtt{R\,e} \to \mathtt{R\,t}$

$\circledast$

$\lambda r.\,\mathsf{spoke}$

$\mathtt{R\,(e \to t)}$

ρ

spoke

$\mathtt{e \to t}$

spoke

Applicatives

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

$$\rho :: a \to F\,a \qquad\qquad \circledast :: F\,(a \to b) \to F\,a \to F\,b$$

...Where ρ 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[1]

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

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

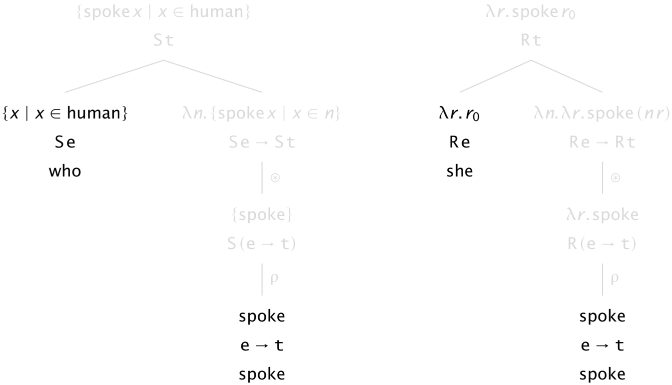
Naturally handled using another applicative functor, for sets::

$$\underbrace{\rho\,x := \{x\}}_{\rho :: a \to \mathtt{S}\,a} \qquad \underbrace{m \circledast n := \{f\,x \mid f \in m, x \in n\}}_{\circledast :: \mathtt{S}\,(a \to b) \to \mathtt{S}\,a \to \mathtt{S}\,b}$$

---

[1] Cf. Hamblin 1973, Shan 2002, Charlow 2014, 2017.

Sample derivation, compared with context-sensitivity



$\{\mathrm{spoke}\,x \mid x \in \mathrm{human}\}$

$\mathtt{St}$

$\{x \mid x \in \mathrm{human}\}$

$\mathtt{Se}$

who

$\lambda n.\,\{\mathrm{spoke}\,x \mid x \in n\}$

$\mathtt{Se} \to \mathtt{St}$

$\circledast$

$\{\mathrm{spoke}\}$

$\mathtt{S}(\mathtt{e} \to \mathtt{t})$

ρ

spoke

$\mathtt{e} \to \mathtt{t}$

spoke

$\lambda r.\,\mathrm{spoke}\,r_0$

$\mathtt{Rt}$

$\lambda r.\,r_0$

$\mathtt{Re}$

she

$\lambda n.\,\lambda r.\,\mathrm{spoke}\,(n\,r)$

$\mathtt{Re} \to \mathtt{Rt}$

$\circledast$

$\lambda r.\,\mathrm{spoke}$

$\mathtt{R}(\mathtt{e} \to \mathtt{t})$

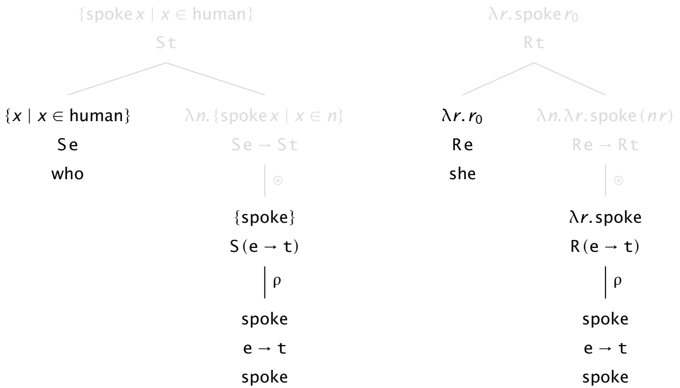
ρ

spoke

$\mathtt{e} \to \mathtt{t}$

spoke

Sample derivation, compared with context-sensitivity



$\{\mathrm{spoke}\,x \mid x \in \mathrm{human}\}$

$\mathtt{S\,t}$

$\{x \mid x \in \mathrm{human}\}$

$\mathtt{S\,e}$

who

$\lambda n.\,\{\mathrm{spoke}\,x \mid x \in n\}$

$\mathtt{S\,e} \to \mathtt{S\,t}$

$\circledast$

$\{\mathrm{spoke}\}$

$\mathtt{S\,(e \to t)}$

ρ

spoke

$\mathtt{e \to t}$

spoke

$\lambda r.\,\mathrm{spoke}\,r_0$

$\mathtt{R\,t}$

$\lambda r.\,r_0$

$\mathtt{R\,e}$

she

$\lambda n.\lambda r.\,\mathrm{spoke}\,(n\,r)$

$\mathtt{R\,e} \to \mathtt{R\,t}$

$\circledast$

$\lambda r.\,\mathrm{spoke}$

$\mathtt{R\,(e \to t)}$

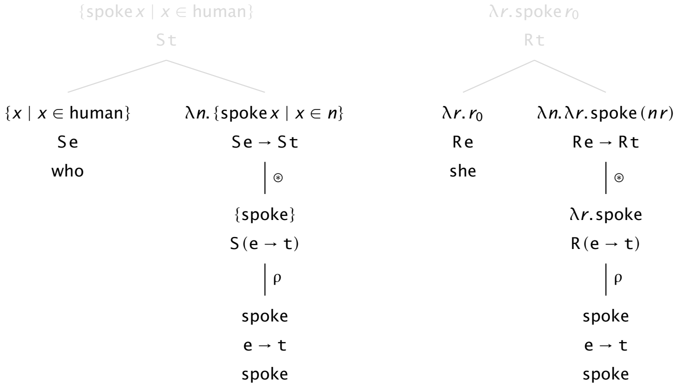
ρ

spoke

$\mathtt{e \to t}$

spoke

Sample derivation, compared with context-sensitivity



$\{\text{spoke}\,x \mid x \in \text{human}\}$

$\mathtt{S\,t}$

$\{x \mid x \in \text{human}\}$

$\mathtt{S\,e}$

who

$\lambda n.\{\text{spoke}\,x \mid x \in n\}$

$\mathtt{S\,e \to S\,t}$

$\circledast$

$\{\text{spoke}\}$

$\mathtt{S\,(e \to t)}$

ρ

spoke

$\mathtt{e \to t}$

spoke

$\lambda r.\,\text{spoke}\,r_0$

$\mathtt{R\,t}$

$\lambda r.\,r_0$

$\mathtt{R\,e}$

she

$\lambda n.\lambda r.\,\text{spoke}\,(n\,r)$

$\mathtt{R\,e \to R\,t}$

$\circledast$

$\lambda r.\,\text{spoke}$

$\mathtt{R\,(e \to t)}$

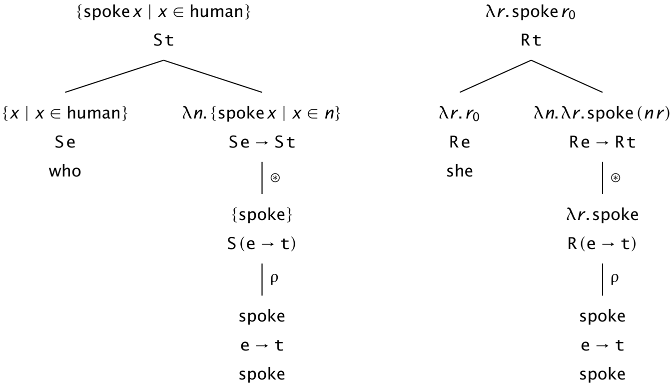
ρ

spoke

$\mathtt{e \to t}$

spoke

Sample derivation, compared with context-sensitivity



$\{\mathsf{spoke}\,x \mid x \in \mathsf{human}\}$

$\mathtt{S\,t}$

$\{x \mid x \in \mathsf{human}\}$

$\mathtt{S\,e}$

who

$\lambda n.\,\{\mathsf{spoke}\,x \mid x \in n\}$

$\mathtt{S\,e} \to \mathtt{S\,t}$

$\circledast$

$\{\mathsf{spoke}\}$

$\mathtt{S}\,(\mathtt{e} \to \mathtt{t})$

ρ

spoke

$\mathtt{e} \to \mathtt{t}$

spoke

$\lambda r.\,\mathsf{spoke}\,r_0$

$\mathtt{R\,t}$

$\lambda r.\,r_0$

$\mathtt{R\,e}$

she

$\lambda n.\lambda r.\,\mathsf{spoke}\,(n\,r)$

$\mathtt{R\,e} \to \mathtt{R\,t}$

$\circledast$

$\lambda r.\,\mathsf{spoke}$

$\mathtt{R}\,(\mathtt{e} \to \mathtt{t})$

ρ

spoke

$\mathtt{e} \to \mathtt{t}$

spoke

Supplementation[2]

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

5. Joe, a linguist, lectured. $\rightsquigarrow$ $(\mathsf{lectured}\,\mathsf{j}, [\mathsf{ling}\,\mathsf{j}]) :: \mathtt{W}\,\mathtt{t}$
6. Joe, a linguist, knows Mary, a philosopher. $\rightsquigarrow$ $(\mathsf{knows}\,\mathsf{m}\,\mathsf{j}, [\mathsf{ling}\,\mathsf{j}, \mathsf{phil}\,\mathsf{m}]) :: \mathtt{W}\,\mathtt{t}$

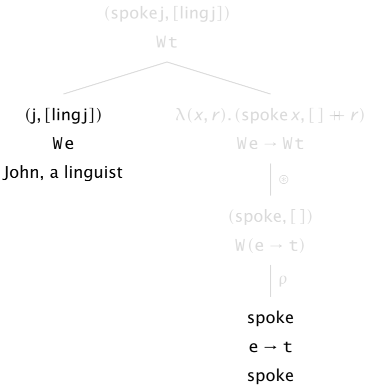
Another example of an applicative functor, for supplements:

$$\underbrace{\rho\, x \coloneqq (x, [\,])}_{\rho :: a \to \mathtt{W}\,a} \qquad \underbrace{(f, l) \circledast (x, r) \coloneqq (f\,x, l +\!\!+\, r)}_{\circledast :: \mathtt{W}\,(a \to b) \to \mathtt{W}\,a \to \mathtt{W}\,b}$$

---

[2] Cf. Potts (2005), Giorgolo & Asudeh (2012), AnderBois, Brasoveanu & Henderson (2015).

Sample derivation: Supplementation



$(\mathsf{spoke}\,\mathsf{j}, [\mathsf{ling}\,\mathsf{j}])$

$\mathtt{W\,t}$

$(\mathsf{j}, [\mathsf{ling}\,\mathsf{j}])$

$\mathtt{W\,e}$

John, a linguist

$\lambda(x, r).\,(\mathsf{spoke}\,x, [\,] \mathbin{+\!\!+} r)$

$\mathtt{W\,e} \to \mathtt{W\,t}$

$\circledast$

$(\mathsf{spoke}, [\,])$

$\mathtt{W}(\mathtt{e} \to \mathtt{t})$

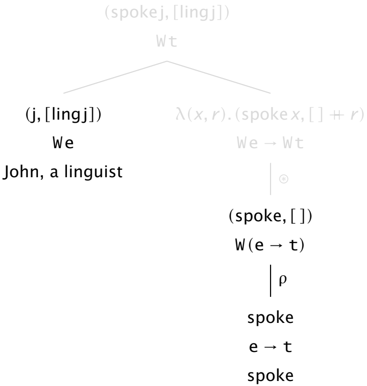
ρ

spoke

$\mathtt{e} \to \mathtt{t}$

spoke

Sample derivation: Supplementation



$(\text{spoke}\,\text{j}, [\text{ling}\,\text{j}])$

$\mathtt{Wt}$

$(\text{j}, [\text{ling}\,\text{j}])$

$\mathtt{We}$

John, a linguist

$\lambda(x, r).\,(\text{spoke}\,x, [\,] +\!\!\!+\, r)$

$\mathtt{We} \to \mathtt{Wt}$

$\circledast$

$(\text{spoke}, [\,])$

$\mathtt{W(e \to t)}$

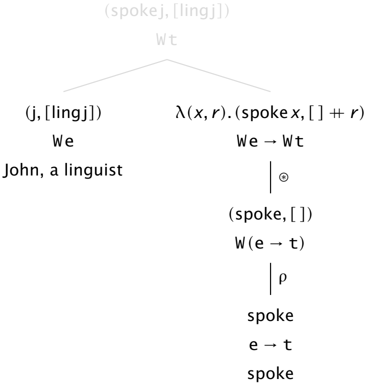
ρ

spoke

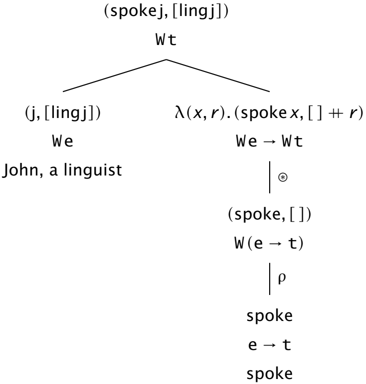
$\mathtt{e \to t}$

spoke

Sample derivation: Supplementation



Sample derivation: Supplementation



$(\text{spoke}\,\text{j}, [\text{ling}\,\text{j}])$

$\mathtt{Wt}$

$(\text{j}, [\text{ling}\,\text{j}])$

$\mathtt{We}$

John, a linguist

$\lambda(x, r).\,(\text{spoke}\,x, [\,] \mathbin{+\!\!+} r)$

$\mathtt{We} \to \mathtt{Wt}$

$\circledast$

$(\text{spoke}, [\,])$

$\mathtt{W(e \to t)}$

ρ

spoke

$\mathtt{e \to t}$

spoke

Scope and continuations

Languages have quantificational expressions, and they take scope:

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

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

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

The relevant enrichment handles expressions with a scope (continuation):[3]

$$\mathsf{C}_r\,a ::= (a \to r) \to r \qquad \forall, \exists :: \mathsf{C}_{\mathtt{t}}\,\mathsf{e} = (\mathsf{e} \to \mathtt{t}) \to \mathtt{t}$$

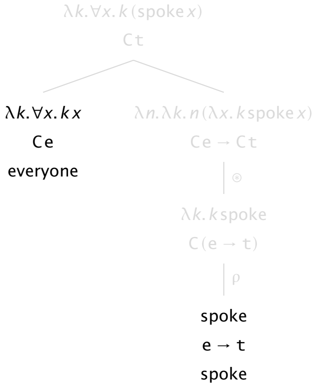
Yet another example of an applicative functor, for scope (continuations):

$$\rho\,x := \lambda k.\,k\,x \qquad m \circledast n := \lambda k.\,m\,(\lambda f.\,n\,(\lambda x.\,k\,(f\,x)))$$

---

[3] Barker (2002), Shan (2002), Shan & Barker (2006), Barker & Shan (2008, 2014), Charlow (2014).

Sample derivation: Scope



$\lambda k.\forall x.\,k\,(\text{spoke}\,x)$

C t

$\lambda k.\forall x.\,k\,x$

C e

everyone

$\lambda n.\lambda k.\,n\,(\lambda x.\,k\,\text{spoke}\,x)$

C e → C t

⊛

$\lambda k.\,k\,\text{spoke}$

C (e → t)

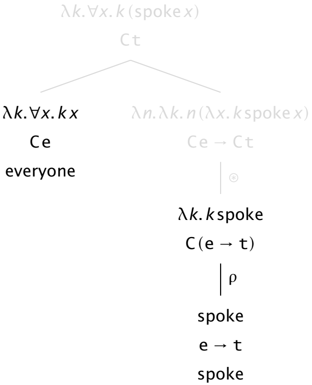
ρ

spoke

e → t

spoke

Sample derivation: Scope



$\lambda k.\forall x.\, k\,(\mathsf{spoke}\, x)$

$\mathtt{C\,t}$

$\lambda k.\forall x.\, k\, x$

$\mathtt{C\,e}$

everyone

$\lambda n.\lambda k.\, n\,(\lambda x.\, k\,\mathsf{spoke}\, x)$

$\mathtt{C\,e} \to \mathtt{C\,t}$

$\circledast$

$\lambda k.\, k\,\mathsf{spoke}$

$\mathtt{C\,(e \to t)}$

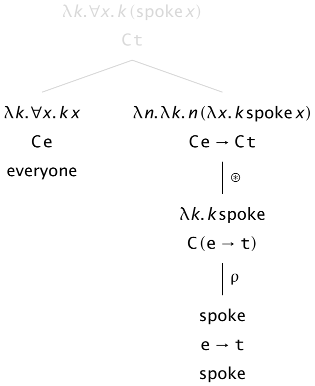
ρ

spoke

$\mathtt{e \to t}$

spoke

Sample derivation: Scope



$\lambda k. \forall x. k\,(\mathsf{spoke}\,x)$

$\mathtt{C}\,\mathtt{t}$

$\lambda k. \forall x. k\,x$

$\mathtt{C}\,\mathtt{e}$

everyone

$\lambda n. \lambda k. n\,(\lambda x. k\,\mathsf{spoke}\,x)$

$\mathtt{C}\,\mathtt{e} \to \mathtt{C}\,\mathtt{t}$

$\circledast$

$\lambda k. k\,\mathsf{spoke}$

$\mathtt{C}\,(\mathtt{e} \to \mathtt{t})$

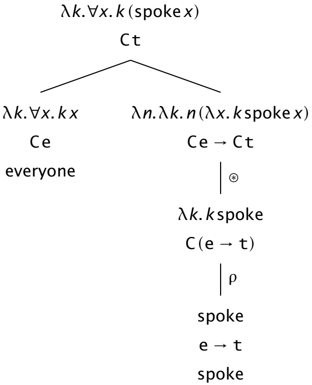
ρ

spoke

$\mathtt{e} \to \mathtt{t}$

spoke

Sample derivation: Scope



Scope alternations via flexibility in ⊛

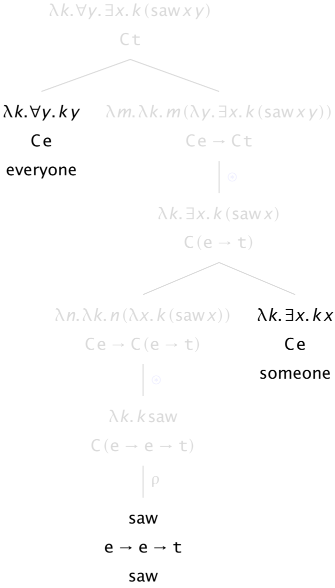
It turns out that the Continuations applicative is *non-commutative* in that it admits two ⊛'s which evaluate their arguments in opposite orders.

$$\rho\, x \coloneqq \lambda k.\, k\, x$$

$$\underbrace{m \circledast n \coloneqq \lambda k.\, m\,(\lambda f.\, n\,(\lambda x.\, k\,(f\, x)))}_{\text{function-first}}$$

$$\underbrace{m \circledast n \coloneqq \lambda k.\, n\,(\lambda x.\, m\,(\lambda f.\, k\,(f\, x)))}_{\text{argument-first}}$$

A couple examples



$\lambda k.\forall y.\exists x.k\,(\mathrm{saw}\,x\,y)$

$\mathrm{C\,t}$

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

$\mathrm{C\,e}$

everyone

$\lambda m.\lambda k.m\,(\lambda y.\exists x.k\,(\mathrm{saw}\,x\,y))$

$\mathrm{C\,e} \to \mathrm{C\,t}$

$\circledast$

$\lambda k.\exists x.k\,(\mathrm{saw}\,x)$

$\mathrm{C\,(e \to t)}$

$\lambda n.\lambda k.n\,(\lambda x.k\,(\mathrm{saw}\,x))$

$\mathrm{C\,e} \to \mathrm{C\,(e \to t)}$

$\circledast$

$\lambda k.k\,\mathrm{saw}$

$\mathrm{C\,(e \to e \to t)}$

ρ

saw

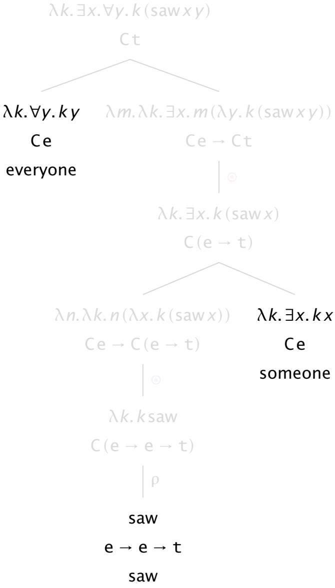
$\mathrm{e \to e \to t}$

saw

$\lambda k.\exists x.k\,x$

$\mathrm{C\,e}$

someone



$\lambda k.\exists x.\forall y.k\,(\mathrm{saw}\,x\,y)$

$\mathrm{C\,t}$

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

$\mathrm{C\,e}$

everyone

$\lambda m.\lambda k.\exists x.m\,(\lambda y.k\,(\mathrm{saw}\,x\,y))$

$\mathrm{C\,e} \to \mathrm{C\,t}$

$\circledast$

$\lambda k.\exists x.k\,(\mathrm{saw}\,x)$

$\mathrm{C\,(e \to t)}$

$\lambda n.\lambda k.n\,(\lambda x.k\,(\mathrm{saw}\,x))$

$\mathrm{C\,e} \to \mathrm{C\,(e \to t)}$

$\circledast$

$\lambda k.k\,\mathrm{saw}$

$\mathrm{C\,(e \to e \to t)}$

ρ

saw

$\mathrm{e \to e \to t}$

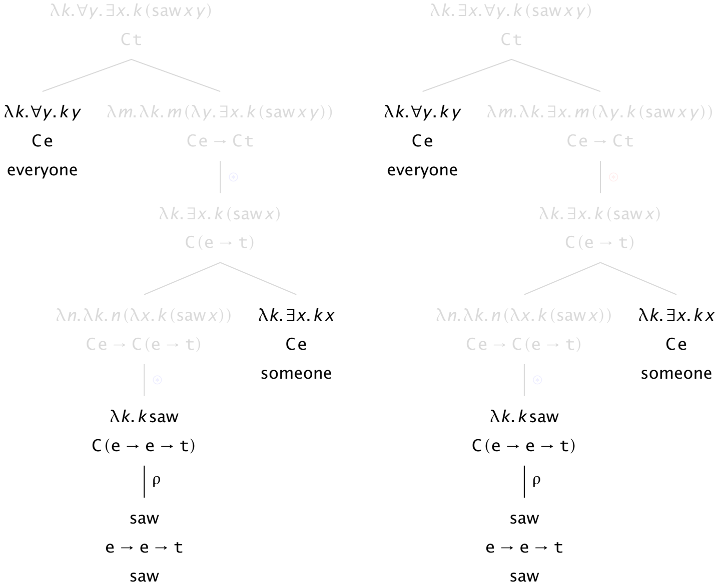
saw

$\lambda k.\exists x.k\,x$

$\mathrm{C\,e}$

someone

A couple examples



$\lambda k.\forall y.\exists x.k\,(\mathrm{saw}\,x\,y)$

$\mathsf{C}\,\mathsf{t}$

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

$\mathsf{C}\,\mathsf{e}$

everyone

$\lambda m.\lambda k.m\,(\lambda y.\exists x.k\,(\mathrm{saw}\,x\,y))$

$\mathsf{C}\,\mathsf{e} \to \mathsf{C}\,\mathsf{t}$

$\circledast$

$\lambda k.\exists x.k\,(\mathrm{saw}\,x)$

$\mathsf{C}\,(\mathsf{e} \to \mathsf{t})$

$\lambda n.\lambda k.n\,(\lambda x.k\,(\mathrm{saw}\,x))$

$\mathsf{C}\,\mathsf{e} \to \mathsf{C}\,(\mathsf{e} \to \mathsf{t})$

$\circledast$

$\lambda k.k\,\mathrm{saw}$

$\mathsf{C}\,(\mathsf{e} \to \mathsf{e} \to \mathsf{t})$

ρ

saw

$\mathsf{e} \to \mathsf{e} \to \mathsf{t}$

saw

$\lambda k.\exists x.k\,x$

$\mathsf{C}\,\mathsf{e}$

someone

$\lambda k.\exists x.\forall y.k\,(\mathrm{saw}\,x\,y)$

$\mathsf{C}\,\mathsf{t}$

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

$\mathsf{C}\,\mathsf{e}$

everyone

$\lambda m.\lambda k.\exists x.m\,(\lambda y.k\,(\mathrm{saw}\,x\,y))$

$\mathsf{C}\,\mathsf{e} \to \mathsf{C}\,\mathsf{t}$

$\circledast$

$\lambda k.\exists x.k\,(\mathrm{saw}\,x)$

$\mathsf{C}\,(\mathsf{e} \to \mathsf{t})$

$\lambda n.\lambda k.n\,(\lambda x.k\,(\mathrm{saw}\,x))$

$\mathsf{C}\,\mathsf{e} \to \mathsf{C}\,(\mathsf{e} \to \mathsf{t})$

$\circledast$

$\lambda k.k\,\mathrm{saw}$

$\mathsf{C}\,(\mathsf{e} \to \mathsf{e} \to \mathsf{t})$

ρ

saw

$\mathsf{e} \to \mathsf{e} \to \mathsf{t}$

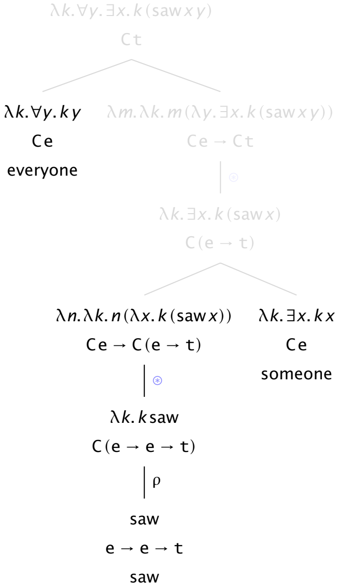
saw

$\lambda k.\exists x.k\,x$

$\mathsf{C}\,\mathsf{e}$

someone

A couple examples



$\lambda k.\forall y.\exists x.k\,(\mathsf{saw}\,x\,y)$

$\mathsf{C\,t}$

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

$\mathsf{C\,e}$

everyone

$\lambda m.\lambda k.m\,(\lambda y.\exists x.k\,(\mathsf{saw}\,x\,y))$

$\mathsf{C\,e} \to \mathsf{C\,t}$

$\circledast$

$\lambda k.\exists x.k\,(\mathsf{saw}\,x)$

$\mathsf{C\,(e \to t)}$

$\lambda n.\lambda k.n\,(\lambda x.k\,(\mathsf{saw}\,x))$

$\mathsf{C\,e} \to \mathsf{C\,(e \to t)}$

$\circledast$

$\lambda k.k\,\mathsf{saw}$

$\mathsf{C\,(e \to e \to t)}$

ρ

saw

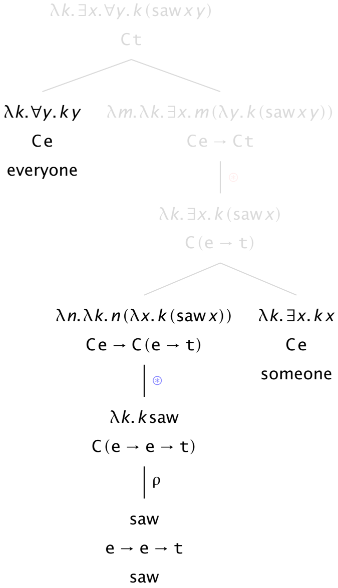
$\mathsf{e \to e \to t}$

saw

$\lambda k.\exists x.k\,x$

$\mathsf{C\,e}$

someone



$\lambda k.\exists x.\forall y.k\,(\mathsf{saw}\,x\,y)$

$\mathsf{C\,t}$

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

$\mathsf{C\,e}$

everyone

$\lambda m.\lambda k.\exists x.m\,(\lambda y.k\,(\mathsf{saw}\,x\,y))$

$\mathsf{C\,e} \to \mathsf{C\,t}$

$\circledast$

$\lambda k.\exists x.k\,(\mathsf{saw}\,x)$

$\mathsf{C\,(e \to t)}$

$\lambda n.\lambda k.n\,(\lambda x.k\,(\mathsf{saw}\,x))$

$\mathsf{C\,e} \to \mathsf{C\,(e \to t)}$

$\circledast$

$\lambda k.k\,\mathsf{saw}$

$\mathsf{C\,(e \to e \to t)}$

ρ

saw

$\mathsf{e \to e \to t}$

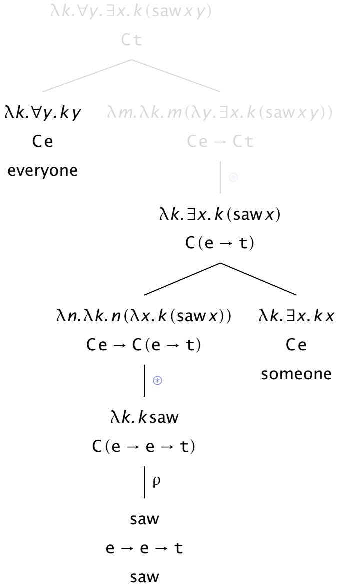
saw

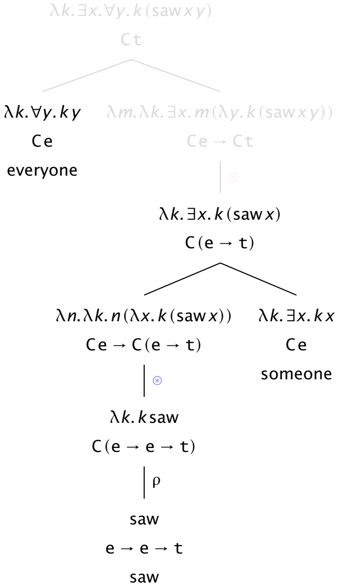
$\lambda k.\exists x.k\,x$

$\mathsf{C\,e}$

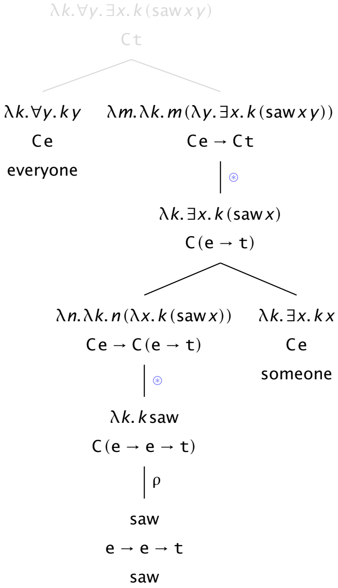
someone

A couple examples





A couple examples



$\lambda k.\forall y.\exists x.k\,(\mathsf{saw}\,x\,y)$

$\mathtt{C\,t}$

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

$\mathtt{C\,e}$

everyone

$\lambda m.\lambda k.m\,(\lambda y.\exists x.k\,(\mathsf{saw}\,x\,y))$

$\mathtt{C\,e} \to \mathtt{C\,t}$

$\circledast$

$\lambda k.\exists x.k\,(\mathsf{saw}\,x)$

$\mathtt{C\,(e \to t)}$

$\lambda n.\lambda k.n\,(\lambda x.k\,(\mathsf{saw}\,x))$

$\mathtt{C\,e} \to \mathtt{C\,(e \to t)}$

$\circledast$

$\lambda k.k\,\mathsf{saw}$

$\mathtt{C\,(e \to e \to t)}$

ρ

saw

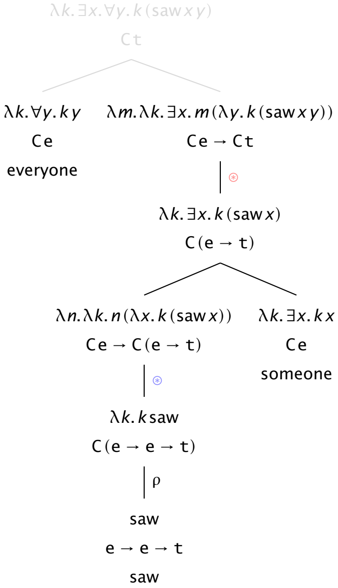
$\mathtt{e \to e \to t}$

saw

$\lambda k.\exists x.k\,x$

$\mathtt{C\,e}$

someone



$\lambda k.\exists x.\forall y.k\,(\mathsf{saw}\,x\,y)$

$\mathtt{C\,t}$

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

$\mathtt{C\,e}$

everyone

$\lambda m.\lambda k.\exists x.m\,(\lambda y.k\,(\mathsf{saw}\,x\,y))$

$\mathtt{C\,e} \to \mathtt{C\,t}$

$\circledast$

$\lambda k.\exists x.k\,(\mathsf{saw}\,x)$

$\mathtt{C\,(e \to t)}$

$\lambda n.\lambda k.n\,(\lambda x.k\,(\mathsf{saw}\,x))$

$\mathtt{C\,e} \to \mathtt{C\,(e \to t)}$

$\circledast$

$\lambda k.k\,\mathsf{saw}$

$\mathtt{C\,(e \to e \to t)}$

ρ

saw

$\mathtt{e \to e \to t}$

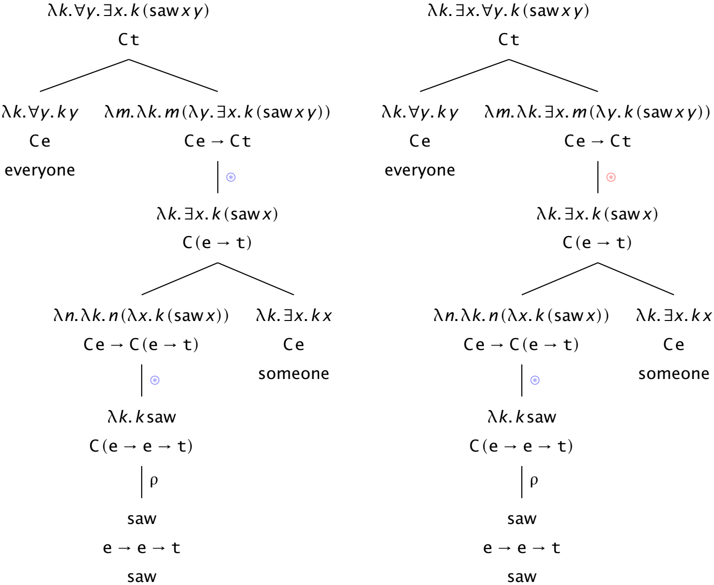
saw

$\lambda k.\exists x.k\,x$

$\mathtt{C\,e}$

someone

A couple examples



$\lambda k.\forall y.\exists x.k\,(\mathsf{saw}\,x\,y)$

$\mathtt{C\,t}$

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

$\mathtt{C\,e}$

everyone

$\lambda m.\lambda k.m\,(\lambda y.\exists x.k\,(\mathsf{saw}\,x\,y))$

$\mathtt{C\,e \to C\,t}$

$\circledast$

$\lambda k.\exists x.k\,(\mathsf{saw}\,x)$

$\mathtt{C\,(e \to t)}$

$\lambda n.\lambda k.n\,(\lambda x.k\,(\mathsf{saw}\,x))$

$\mathtt{C\,e \to C\,(e \to t)}$

$\circledast$

$\lambda k.k\,\mathsf{saw}$

$\mathtt{C\,(e \to e \to t)}$

ρ

saw

$\mathtt{e \to e \to t}$

saw

$\lambda k.\exists x.k\,x$

$\mathtt{C\,e}$

someone

$\lambda k.\exists x.\forall y.k\,(\mathsf{saw}\,x\,y)$

$\mathtt{C\,t}$

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

$\mathtt{C\,e}$

everyone

$\lambda m.\lambda k.\exists x.m\,(\lambda y.k\,(\mathsf{saw}\,x\,y))$

$\mathtt{C\,e \to C\,t}$

$\circledast$

$\lambda k.\exists x.k\,(\mathsf{saw}\,x)$

$\mathtt{C\,(e \to t)}$

$\lambda n.\lambda k.n\,(\lambda x.k\,(\mathsf{saw}\,x))$

$\mathtt{C\,e \to C\,(e \to t)}$

$\circledast$

$\lambda k.k\,\mathsf{saw}$

$\mathtt{C\,(e \to e \to t)}$

ρ

saw

$\mathtt{e \to e \to t}$

saw

$\lambda k.\exists x.k\,x$

$\mathtt{C\,e}$

someone

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)
```

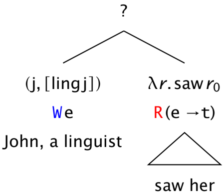
Similar to enriching $[\![\cdot]\!]$. Another possibility, more closely related to the strategy we're using here, is having applicative combinators in the object language.

- (That's how Haskell programmers roll.)

Reading and writing

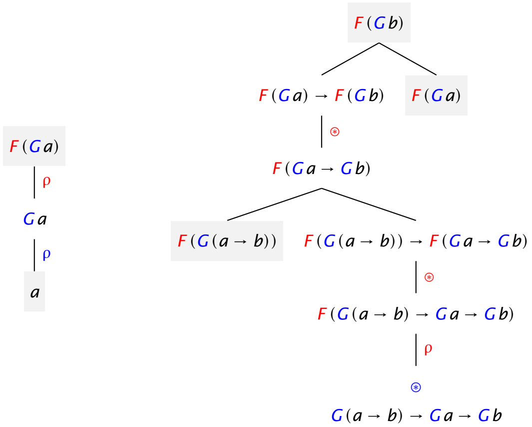
Simultaneous effects

How to combine expressions from *different* effect regimes?



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

Applicative functors automatically compose



$F\,(G\,a)$

ρ

$G\,a$

ρ

a

$F\,(G\,b)$

$F\,(G\,a) \to F\,(G\,b)$

$F\,(G\,a)$

$\circledast$

$F\,(G\,a \to G\,b)$

$F\,(G\,(a \to b))$

$F\,(G\,(a \to b)) \to F\,(G\,a \to G\,b)$

$\circledast$

$F\,(G\,(a \to b) \to G\,a \to G\,b)$

ρ

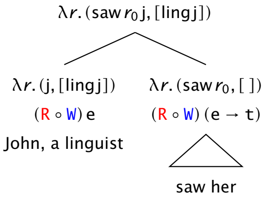
$\circledast$

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

Composition with composition

Here's what we get for the composition of $\mathtt{R}$ and $\mathtt{W}$, $(\mathtt{R} \circ \mathtt{W})\,\mathtt{a} = \mathtt{r} \to (\mathtt{a}, [\mathtt{t}])$:

$$\rho\, x := \lambda r.\,(x, [\,]) \qquad m \circledast n := \lambda r.\,(f\,x, j \mathbin{+\!\!+} k) \textbf{ where } \begin{aligned}(f, j) &:= m\,r\\ (x, k) &:= n\,r\end{aligned}$$



$\lambda r.\,(\mathsf{saw}\,r_0\,\mathsf{j}, [\mathsf{ling}\,\mathsf{j}])$

$\lambda r.\,(\mathsf{j}, [\mathsf{ling}\,\mathsf{j}])$

$(\mathtt{R} \circ \mathtt{W})\,\mathtt{e}$

John, a linguist

$\lambda r.\,(\mathsf{saw}\,r_0, [\,])$

$(\mathtt{R} \circ \mathtt{W})\,(\mathtt{e} \to \mathtt{t})$

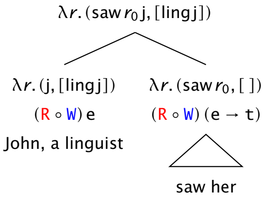
saw her

$\mathtt{R} \circ \mathtt{W}$ also implies ways to lift $\mathtt{R}\,a$ and $\mathtt{W}\,a$ into $(\mathtt{R} \circ \mathtt{W})\,a$. Exercise: find them!

Taking the reverse composition

Here’s what we get for the *reverse* composition of $\mathtt{R}$ and $\mathtt{W}$, $(\mathtt{R} \circ \mathtt{W})\,\mathtt{a} = \mathtt{r} \to (\mathtt{a}, [\mathtt{t}])$:

$$\rho\, x := \lambda r.\,(x, [\,]) \qquad \begin{aligned} m \circledast n := \lambda r.\,(f\,x, j +\!\!+\, k) \textbf{ where } (f, j) &:= m\,r \\ (x, k) &:= n\,r \end{aligned}$$



$\mathtt{R} \circ \mathtt{W}$ also implies ways to lift $\mathtt{R}\,a$ and $\mathtt{W}\,a$ into $(\mathtt{R} \circ \mathtt{W})\,a$. Exercise: find them!

Some more composed applicatives[4]

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

$$\rho\, x \coloneqq \lambda r.\, \{x\} \qquad m \circledast n \coloneqq \lambda r.\, \{f\, x \mid f \in m\, r, x \in n\, r\}$$
$$= \rho\, (\rho\, x) \qquad\qquad = (\rho \circledast) \circledast m \circledast n$$

This corresponds to what is standardly called Alternative Semantics.

And here, for S $\circ$ R:

$$\rho\, x \coloneqq \{\lambda r.\, x\} \qquad m \circledast n \coloneqq \{\lambda r.\, f\, r\, (x\, r) \mid f \in m, x \in n\}$$
$$= \rho\, (\rho\, x) \qquad\qquad = (\rho \circledast) \circledast m \circledast n$$

---

[4] 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.

8. $\underbrace{\text{Polly}}_{\text{Write}}$ walked in the park. $\underbrace{\text{She}}_{\text{Read}}$ whistled.

Composing reading and writing actions

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

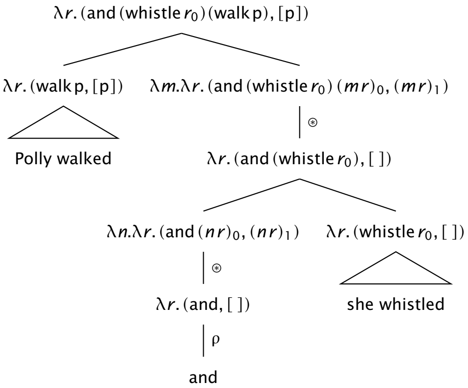
$$(\mathtt{R} \circ \mathtt{W})\,\mathtt{a} ::= \mathtt{r} \to (\mathtt{a}, [\mathtt{e}])$$

And the corresponding ρand ⊛ operations again:

$$\rho x := \lambda r.\,(x, [\,]) \qquad m \circledast n := \lambda r.\,(f\,x, j \mathbin{+\!\!+} k)\ \textbf{where}\ \begin{array}{l} (f, j)\ := m\,r \\ (x, k)\ := n\,r \end{array}$$

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

Failure to communicate



$\lambda r.\,(\text{and}\,(\text{whistle}\,r_0)\,(\text{walk}\,\text{p}),\,[\text{p}])$

$\lambda r.\,(\text{walk}\,\text{p},\,[\text{p}])$

Polly walked

$\lambda m.\lambda r.\,(\text{and}\,(\text{whistle}\,r_0)\,(m\,r)_0,\,(m\,r)_1\,)$

$\circledast$

$\lambda r.\,(\text{and}\,(\text{whistle}\,r_0),\,[\,])$

$\lambda n.\lambda r.\,(\text{and}\,(n\,r)_0,\,(n\,r)_1\,)$

$\circledast$

$\lambda r.\,(\text{and},\,[\,])$

ρ

and

$\lambda r.\,(\text{whistle}\,r_0,\,[\,])$

she whistled

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.

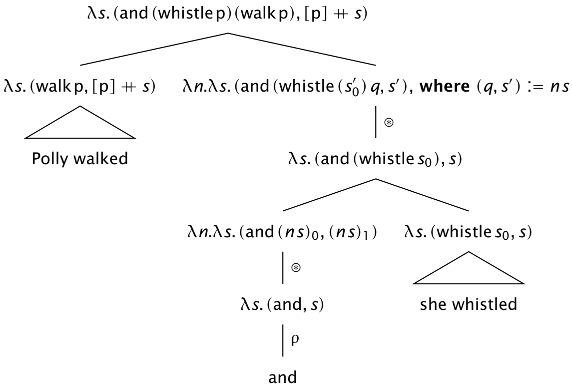
The State applicative: $\mathsf{ST}\,a ::= \mathsf{s} \to (\mathsf{a}, \mathsf{s})$

$$\rho x := \lambda s.\,(x, s) \qquad m \circledast n := \lambda s.\,(f\,x, s'')\ \textbf{where}\ \begin{aligned}(f, s') &:= m\,s\\ (x, s'') &:= n\,s'\end{aligned}$$

$$\rho x := \lambda r.\,(x, [\,]) \qquad m \circledast n := \lambda r.\,(f\,x, j +\!\!+\, k)\ \textbf{where}\ \begin{aligned}(f, j) &:= m\,r\\ (x, k) &:= n\,r\end{aligned}$$

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

Successful communication



$\lambda s.\,(\text{and}\,(\text{whistle}\,\text{p})\,(\text{walk}\,\text{p}), [\text{p}] \mathbin{+\!\!+} s)$

$\lambda s.\,(\text{walk}\,\text{p}, [\text{p}] \mathbin{+\!\!+} s)$

Polly walked

$\lambda n.\lambda s.\,(\text{and}\,(\text{whistle}\,(s'_0)\,q, s'),\ \textbf{where}\ (q, s') := n\,s$

$\circledast$

$\lambda s.\,(\text{and}\,(\text{whistle}\,s_0), s)$

$\lambda n.\lambda s.\,(\text{and}\,(n\,s)_0, (n\,s)_1)$

$\lambda s.\,(\text{whistle}\,s_0, s)$

$\circledast$

$\lambda s.\,(\text{and}, s)$

she whistled

ρ

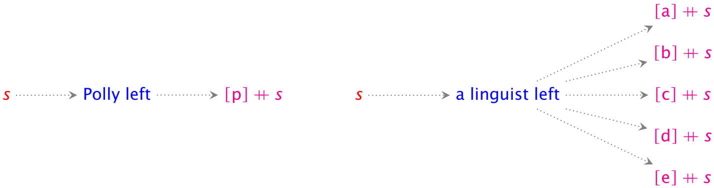
and

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

Indefinites[5]

True dynamic effects combine reading/writing with *nondeterminism*:

9. Polly walked in the park. She whistled.
10. A linguist walked in the park. She whistled.



s → Polly left → $[p] +\!\!+ s$

s → a linguist left → $[a] +\!\!+ s$, $[b] +\!\!+ s$, $[c] +\!\!+ s$, $[d] +\!\!+ s$, $[e] +\!\!+ s$

---

[5] Heim (1982), Barwise (1987), Rooth (1987), Groenendijk & Stokhof (1991), Muskens (1996), etc.

Nondeterministic compositions with ST

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

$\mathsf{ST} \circ \mathsf{S}$:

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

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

$\mathsf{S} \circ \mathsf{ST}$:

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

$$\begin{aligned} m \circledast n := \lambda s.\,(\{f\,x \mid f \in F, x \in X\}, s'')\ \textbf{where},\ (F, s') &:= m\,s \\ (X, s'') &:= n\,s' \end{aligned}$$

Problems with these compositions

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

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

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

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

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

$$[\![\text{a book she read}]\!] \neq \left\{\lambda s.\,(x, [x] +\!\!+ s) \mid \text{book}\,x,\ \text{read}\,\underline{\textbf{she}}\,x\right\}$$

Nondeterministic state applicative

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

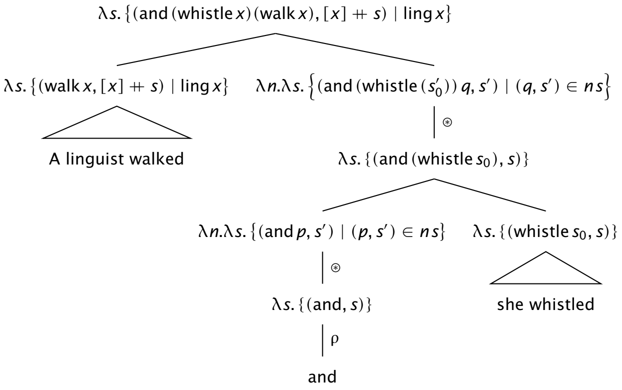
$$\mathrm{D}\,a ::= \mathrm{s} \to \{(\mathrm{a}, \mathrm{s})\}$$

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

$$[\![\text{a linguist}]\!] := \lambda s.\, \{(x, [x] \mathbin{+\!\!+} s) \mid \text{ling}\, x\}$$

$$[\![\text{a book she read}]\!] := \lambda s.\, \{(x, [x] \mathbin{+\!\!+} s) \mid \text{book}\, x,\ \text{read}\, s_0\, x\}$$

Dynamics in action



$$\lambda s.\,\{(\mathsf{and}\,(\mathsf{whistle}\,x)\,(\mathsf{walk}\,x), [x] \mathbin{+\!\!+} s) \mid \mathsf{ling}\,x\}$$

$$\lambda s.\,\{(\mathsf{walk}\,x, [x] \mathbin{+\!\!+} s) \mid \mathsf{ling}\,x\}$$

A linguist walked

$$\lambda n.\lambda s.\,\left\{(\mathsf{and}\,(\mathsf{whistle}\,(s'_0))\,q, s') \mid (q, s') \in n\,s\right\}$$

$$\circledast$$

$$\lambda s.\,\{(\mathsf{and}\,(\mathsf{whistle}\,s_0), s)\}$$

$$\lambda n.\lambda s.\,\{(\mathsf{and}\,p, s') \mid (p, s') \in n\,s\}$$

$$\circledast$$

$$\lambda s.\,\{(\mathsf{and}, s)\}$$

$$\rho$$

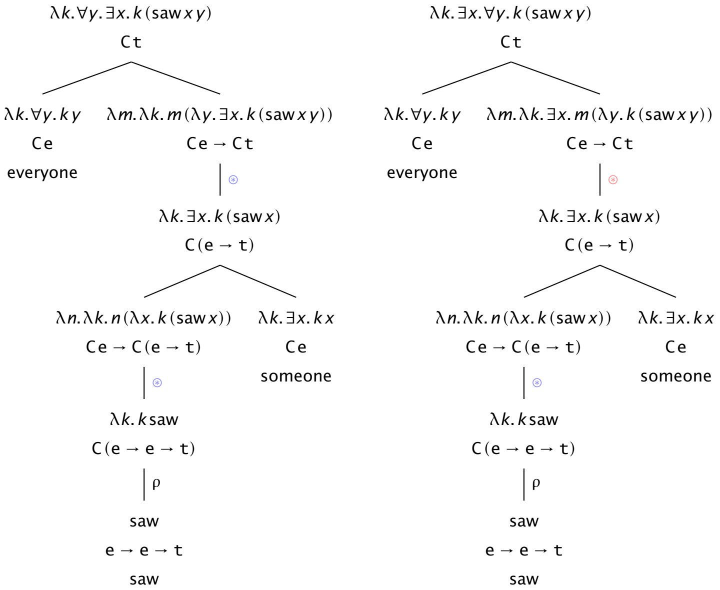
and

$$\lambda s.\,\{(\mathsf{whistle}\,s_0, s)\}$$

she whistled

Scope and monads

Scope interactions, refresher



$\lambda k.\forall y.\exists x.k\,(\mathsf{saw}\,x\,y)$
$\mathsf{C}\,\mathsf{t}$

$\lambda k.\forall y.k\,y$
$\mathsf{C}\,\mathsf{e}$
everyone

$\lambda m.\lambda k.m\,(\lambda y.\exists x.k\,(\mathsf{saw}\,x\,y))$
$\mathsf{C}\,\mathsf{e} \to \mathsf{C}\,\mathsf{t}$
$\circledast$
$\lambda k.\exists x.k\,(\mathsf{saw}\,x)$
$\mathsf{C}\,(\mathsf{e} \to \mathsf{t})$

$\lambda n.\lambda k.n\,(\lambda x.k\,(\mathsf{saw}\,x))$
$\mathsf{C}\,\mathsf{e} \to \mathsf{C}\,(\mathsf{e} \to \mathsf{t})$
$\circledast$
$\lambda k.k\,\mathsf{saw}$
$\mathsf{C}\,(\mathsf{e} \to \mathsf{e} \to \mathsf{t})$
ρ
saw
$\mathsf{e} \to \mathsf{e} \to \mathsf{t}$
saw

$\lambda k.\exists x.k\,x$
$\mathsf{C}\,\mathsf{e}$
someone

$\lambda k.\exists x.\forall y.k\,(\mathsf{saw}\,x\,y)$
$\mathsf{C}\,\mathsf{t}$

$\lambda k.\forall y.k\,y$
$\mathsf{C}\,\mathsf{e}$
everyone

$\lambda m.\lambda k.\exists x.m\,(\lambda y.k\,(\mathsf{saw}\,x\,y))$
$\mathsf{C}\,\mathsf{e} \to \mathsf{C}\,\mathsf{t}$
$\circledast$
$\lambda k.\exists x.k\,(\mathsf{saw}\,x)$
$\mathsf{C}\,(\mathsf{e} \to \mathsf{t})$

$\lambda n.\lambda k.n\,(\lambda x.k\,(\mathsf{saw}\,x))$
$\mathsf{C}\,\mathsf{e} \to \mathsf{C}\,(\mathsf{e} \to \mathsf{t})$
$\circledast$
$\lambda k.k\,\mathsf{saw}$
$\mathsf{C}\,(\mathsf{e} \to \mathsf{e} \to \mathsf{t})$
ρ
saw
$\mathsf{e} \to \mathsf{e} \to \mathsf{t}$
saw

$\lambda k.\exists x.k\,x$
$\mathsf{C}\,\mathsf{e}$
someone

Too many indefinites

These derivations assumed ‘a linguist’ was a generalized quantifier: $\mathtt{C\,e}$

But in the dynamic section, we assumed ‘a linguist’ was a nondeterministic state modifier: $\mathtt{D\,e}$.

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 — $\mathsf{C}_{\mathtt{t}}\,e = (\mathtt{e} \to \mathtt{t}) \to \mathtt{t}$ — as it does for “dynamic” GQs — $\mathsf{C}_{\mathtt{Dt}}\,\mathtt{e} = (\mathtt{e} \to \mathtt{D\,t}) \to \mathtt{D\,t}$

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

$$\text{evOne} :: \mathsf{C}_{\mathtt{Dt}}\,\mathtt{e} = (\mathtt{e} \to \mathtt{D\,t}) \to \mathtt{D\,t}$$

But how are indefinites, type $\mathtt{D\,e}$, supposed to scopally interact with it?

Flipping $\circledast :: \mathtt{D\,(e \to t)} \to \mathtt{D\,e} \to \mathtt{D\,t}$ and applying it to aLing $:: \mathtt{D\,e}$ gives:

$$(\circledast\ \text{aLing}) :: \mathtt{D\,(e \to t)} \to \mathtt{D\,t}$$

So close

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

$$(\mathtt{e} \to \mathtt{D}\,\mathtt{t}) \to \mathtt{D}\,\mathtt{e} \to \mathtt{D}\,\mathtt{t}$$

Actual type, as a reminder:

$$\mathtt{D}\,(\mathtt{e} \to \mathtt{t}) \to \mathtt{D}\,\mathtt{e} \to \mathtt{D}\,\mathtt{t}$$

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:

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

Categorically

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

$$\odot :: F\,a \to (a \to b) \to F\,b \qquad \mu :: F\,(F\,a) \to F\,a$$

in view of the fact that $\mu\,(f \odot m) = m \gg= f$.[6]

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

---

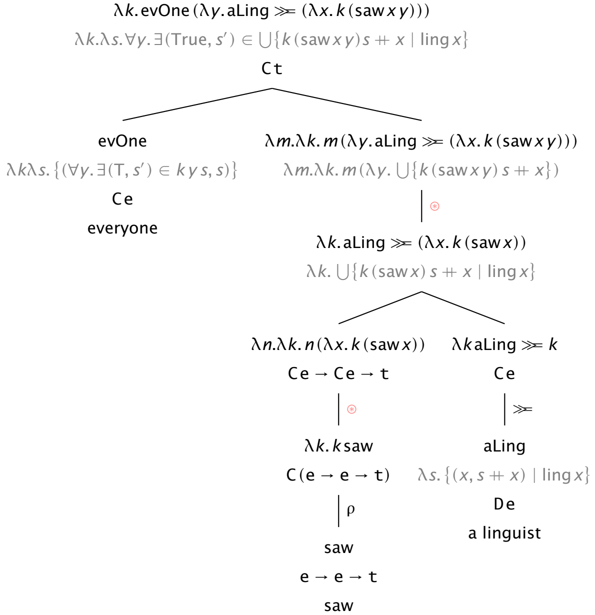
[6] Mere functors can, like monads, interact with continuations, but require the Indexed Continuations applicative. See Shan & Barker (2006), Barker & Shan (2014).

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 $\gg=$



$\lambda k.\,\mathsf{evOne}\,(\lambda y.\,\mathsf{aLing} \gg= (\lambda x.\,k\,(\mathsf{saw}\,x\,y)))$

$\lambda k.\lambda s.\forall y.\exists(\mathsf{True}, s') \in \bigcup\{k\,(\mathsf{saw}\,x\,y)\,s \mathbin{+\!\!+} x \mid \mathsf{ling}\,x\}$

$\mathtt{Ct}$

evOne

$\lambda k \lambda s.\,\{(\forall y.\exists(\mathsf{T}, s') \in k\,y\,s, s)\}$

$\mathtt{Ce}$

everyone

$\lambda m.\lambda k.\,m\,(\lambda y.\,\mathsf{aLing} \gg= (\lambda x.\,k\,(\mathsf{saw}\,x\,y)))$

$\lambda m.\lambda k.\,m\,(\lambda y.\,\bigcup\{k\,(\mathsf{saw}\,x\,y)\,s \mathbin{+\!\!+} x\})$

$\circledast$

$\lambda k.\,\mathsf{aLing} \gg= (\lambda x.\,k\,(\mathsf{saw}\,x))$

$\lambda k.\,\bigcup\{k\,(\mathsf{saw}\,x)\,s \mathbin{+\!\!+} x \mid \mathsf{ling}\,x\}$

$\lambda n.\lambda k.\,n\,(\lambda x.\,k\,(\mathsf{saw}\,x))$

$\mathtt{Ce} \to \mathtt{Ce} \to \mathtt{t}$

$\circledast$

$\lambda k.\,k\,\mathsf{saw}$

$\mathtt{C}\,(\mathtt{e} \to \mathtt{e} \to \mathtt{t})$

ρ

saw

$\mathtt{e} \to \mathtt{e} \to \mathtt{t}$

saw

$\lambda k\,\mathsf{aLing} \gg= k$

$\mathtt{Ce}$

$\gg=$

aLing

$\lambda s.\,\{(x, s \mathbin{+\!\!+} x) \mid \mathsf{ling}\,x\}$

$\mathtt{De}$

a linguist

Scope ambiguity at the end of the day

Statically:

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

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

Dynamically:

$$\mathrm{evOne}\,(\lambda x.\,\mathrm{aLing} \ggcurly \lambda y.\,\eta\,(\mathrm{saw}\,y\,x))$$

$$\mathrm{aLing} \ggcurly \lambda y.\,\mathrm{evOne}\,(\lambda x.\,\eta\,(\mathrm{saw}\,y\,x))$$

Semantic primitives?

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

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

Read (aka R) and `Write` are **adjoint functors** (`Write` $\dashv$ Read). In fact, Read-ing and `Write`-ing are adjoint *in virtue of the curry-uncurry isomorphisms*:

$$L\,a \to b \simeq a \to R\,b$$
$$\texttt{Write}\,a \to b \simeq a \to \text{Read}\,b$$
$$(a, \text{R}) \to b \simeq a \to \text{R} \to b$$

$L \dashv R$ 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., rnon-determinism).[7]

---

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

AnderBois, Scott, Adrian Brasoveanu & Robert Henderson. 2015. At-issue proposals and appositive impositions in discourse. *Journal of Semantics* 32(1). 93–138. https://doi.org/10.1093/jos/fft014.

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.

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://mitwpl.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`.

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`.

Shan, Chung-chieh. 2001. A variable-free dynamic semantics. In Robert van Rooy & 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`.